UNIVERSITY OF KWAZULU-NATAL

Factors influencing software developers' use of pair programming in an agile software development methodology environment

By
Prashika Dhoodhanath
211503263

A dissertation submitted in fulfilment of the requirements for the degree of

Master of Commerce

School of Management, Information Technology and Governance

College of Law and Management Studies

Supervisor: Ms Rosemary Quilling

2018

Declaration

I Prashika Dhoodhanath declare that,

i. The research reported in this dissertation/thesis, except where otherwise indicated, is my original research.

ii. This dissertation/thesis has not been submitted for any degree or examination at any other university.

iii. This dissertation/thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

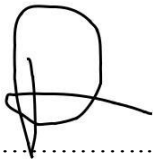
iv. This dissertation/thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

   a. Their words have been re-written but the general information attributed to them has been referenced;

   b. Where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

v. Where I have reproduced a publication of which I am author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

vi. This dissertation/thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation/thesis and in the References sections.

Signed: 

Acknowledgements

My dear parents, Mr Roy and Mrs Sheritha Dhoodhanath, this would not have been possible without you two! Thank you mummy for continuously bringing me back to focus and motivating me especially during my weakest times. Thank you daddy for always supporting and sharing your words of wisdom…thank you both for the eternal support and love! Thank you to my brother, Shevan for supporting me through each and every step and sister, Shekara, for the motivation. Love you all so so very much!

My sincere gratitude Ms Rosemary Quilling, for constantly motivating me, guiding me in the best path possible and providing me with such proficient services of a supervisor, much much appreciated!

And lastly thank you dear God, for all your blessings!

Abstract

IT has been growing rapidly through the years and the IT solutions which are required are no simpler. Industries want IT solutions to be flexible enough to accommodate spikes in demand and to produce outcomes as soon as possible. Therefore, the adoption of agile methodologies has been increasing. extreme programming (XP) has been the most common agile methodology adopted since 2004. Industries have struggled to make the transition from a traditional approach to agile; as there are many opposing principles: traditional methodologies drive individual programming, whereas agile drives team collaboration when developing software. However, the benefits realised from XP grew as companies noticed that teams built strong relationships, software was delivered faster and errors in code were minimal. Pair programming (an XP practice) is the least used XP programming practice. This is in spite of studies conducted in North California in the years 2010 - 2017, which noted that pair programming, when used, provides numerous benefits to both staff and company. Some of the benefits included improving productivity, reducing time spent on delivery; increasing the sharing of knowledge and strengthening teams' morale. This challenges the gap between the adoption of pair programming (which is low) and agile (which is popular). Therefore this study was undertaken to understand the phenomena that influence the adoption of pair programming in agile software development companies.

The results of this case study show that software developers have a positive attitude towards using pair programming. Their senior staff and peers encourage the use of pair programming as the company provides enough hardware and tools to accommodate the needs of pair programming. However, it was indicated by both senior and junior staff that there is reluctance by juniors to voice their opinions. The personality mix sometimes impacts the use of pair programming; for instance, introverts may not want to communicate and an extrovert may be too overpowering in a pair programming environment. However, pair programming is confirmed as a mentoring tool; to help skills development and the sharing of knowledge. In addition, pair programming is noted as more beneficial for complex tasks. Due to the constant engagement required during pair programming, the developers noted it is sometimes draining and therefore suggest regular breaks and switching of roles to maintain the synergy. Overall, pair programming is recommended for future and current use as it produces higher quality code, improves productivity, assists in sharing of knowledge and boosts the confidence and skills of those less experienced.

Table of Contents

Definition of Terms

Agile software development: This is a set of principles for software development that consists of values that underpin solutions to evolve through collaboration. This is procured by iterative, adaptive, continuous and early delivery (Lindstrom & Jeffries, 2004).

Pair programming: This is an agile software development technique that belongs to extreme programming. This technique involves two programmers working together at one computer, solving a task on hand (Williams, 2010).

Software development: This is a process for programming that creates, fixes or maintains software products. Programming involves writing/typing code (Williams, 2010).

Software developers/ developers: These are individuals who conduct the software development i.e. write/type code (Williams, 2010).

TBP (Theory of Planned Behaviour): This is a theory that underpins intention and behaviour of individual's. This theory's aim is to improve the predictability of human behaviour (Icek, 2006).

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background and Context

> *"Pair programming, two programmers collaborating on design, coding and testing, has been a controversial focus of interest as Agile Software Development continues to grow in popularity both among academics and practitioners. As a result of the many investigations into the effectiveness of pair programming in the last decade, many have come to realize that there are many hard-to-control factors in pair programming in particular and in empirical software engineering in general. Because of these factors, the results of many pair programming experiments are not easy to replicate and the relative productivity of pair and solo programming are still not fully understood." - (Kim, Barnes, Chan, & Keith, 2010, p. 143)*

Software development companies currently struggle with the frequent changes in business needs and demands for faster delivery. The traditional approach to software development cannot cope with the ever-changing needs, since within traditional software development; the requirements are locked into the specified period of delivery. This approach is not flexible enough to easily accommodate changes which could have a ripple effect that either pushes delivery to a later time or prevents delivery at all (Cockburn & Williams, 2000). The traditional approach commonly assigns activities as individual tasks. The common perception was that having more than one individual working on a task was a waste of time and a duplication of effort, which would increase the time spent on a task and allows some individuals to contribute less effort as they sponge off others. However, with the rapidly and constantly evolving technology environment, it was becoming impossible to develop high quality software within a limited time using a traditional approach (Kim et al., 2010). Hence, the introduction of agile within software development has been largely adopted due to the savings in time it offers.

Agile software development is about being able to deliver software that can accommodate changes with continuous integrations and iterations so that the software is delivered faster. The agile methodology is embodied by the agile manifesto that is articulated by the 12 agile principles (Doyle, Williams, & Cohn, 2014). The agile manifesto drives a people-centric approach that allows software to be delivered by a team that constantly reflects on and improves delivery. XP is the most commonly practiced agile method within software development industries. One of the least practiced methods is pair programming (Doyle et al., 2014).

Pair programming has many of the benefits of agile when applied to software development. Pair programming is a method that typically involves two software developers sitting in front of one terminal working at a task. There are two roles performed simultaneously: a driver writes the code and a navigator checks/reviews the code being written. One of the benefits is higher quality code as two individuals are constantly reviewing and correcting the code being written. Additionally, the time taken to deliver the code is faster as testing and defect time is reduced. Also, the software developers

are happier when spending most time with a partner and being able to leverage off each other's skills, which improves team work and knowledge transfer (Kim et al., 2010).

This paper is a study of the factors that influence the practice of pair programming within an agile software development industry. The goal of this study is to research the way software developers' program and to engage with them to understand the phenomenon. Previous studies suggest that there are many benefits from using pair programming for software development, hence in this paper the researcher will further analyse the elements that affect the adoption of pair programming, especially in an agile driven organisation.

1.2 Problem Statement

The adoption of agile in software development has increased rapidly. As noted in the 2013 CHAOS report: "In the last 10 years, 45% of agile projects were less than $1 million in labour cost. In contrast, only 14% of waterfall projects were less than $1 million in labour cost" (Standish, 2013, p. 25). The biggest reason for agile being more successful and productive than the traditional waterfall approach is due to the ability of the methodology to break up work in smaller chunks that allow greater focus and faster delivery. Agile promotes interaction among team members aiming for a mutual understanding. One agile practice that belongs to XP is pair programming. As noted in the CHAOS report, "in the XP process developers work in pairs. Often these pairs move around to help create and spread expertise and to generate a sense of a greater team. It is also healthy to periodically evaluate how well team members are working together. Communication is the key to success with any team, and team building requires the active participation and communication with every team member" (Standish, 2013, p. 18). Pair programming is a core practice of XP and noted to be the most aligned to the agile principles (Doyle et al., 2014). Due to the nature of pair programming, a highly interactive approach that involves frequent communication and understanding is used (Williams, 2010).

The benefits derived by using pair programming are those that can aid a company to improve quality, productivity and team work. Pair programming is seen to improve software developers on an individual level through the sharing of knowledge through the continuous interaction and active participation involved when using pair programming. Since pair programming induces each individual to contribute his/her portion of expertise/experiences to a task, this leads to an outcome of diversity and ingenuousness. Besides pair programming assisting individual growth, it has also been seen to greatly increase productivity and quality of work. In some instances it was noted that pair programming was able to also reduce the time spent on tasks (Fu et al., 2017).

However, previous studies had noted that the adoption of pair programming, especially in software development, is low and is in fact the least adopted of agile practices (Doyle et al., 2014). Pair programming generates dynamics that differ vastly from the norm, known as "the traditional/solo

programming method". This is one of the major reasons for the lack of pair programming adoption (Kim et al., 2010).

There is a mismatch between the actual adoption of pair programming by software developers within agile software development and the literature-articulated benefits derived from using pair programming. While the benefits of pair programming are positive, it is actually not adopted much, especially by software development industries (Doyle et al., 2014). Therefore the adoption of pair programming by software practitioners are so low, there is not much empirical evidence found on the usage rates of pair programming within software development industries. This study will explore the factors that influence the adoption of pair programming within software development industries. Since the adoption rate is low the study will focus on the continued use of pair programming after adoption. As mentioned above, previous studies strongly emphasise the inherent benefits of pair programming. However, most experiments were conducted in an educational setting (Williams, Kessler, Cunningham, & Jeffries, 2000).

While pair programming is a popular topic in educational research studies, it is seldom adopted within software development industries. This study will research the factors that influence the use of pair programming, particularly from the perspectives of agile software development industries.

1.3 Research Questions

The intention of this study is to understand the phenomenon of pair programming in an agile software development industry by considering the factors suggested by the Theory of Planned Behaviour. The overarching research question is thus: What influence's software developer's use of pair programming within software development industries. This is deconstructed into four research questions using the study's conceptual framework variables namely: attitude, subjective norms and perceived behavioural control. There will be four main research questions that will be addressed in this study. The research questions are as follows:

1. How do attitudes correlate to software developers' use of pair programming within an agile software development methodology?
2. How do subjective norms correlate to software developers' use of pair programming within an agile software development methodology?
3. How does perceived behavioural control correlate to software developers' use of pair programming within an agile software development methodology?
4. How does intention correlate to software developers' use of pair programming within an agile software development methodology?

1.4 Objectives

The objectives of this study are:

- to determine software developers' *use of pair programming* within an agile software development methodology;
- to assess the role of *attitudes* towards using pair programming within agile software development by determining:
    - the attitudes of software developers towards development when using pair programming within agile software development,
    - the attitudes of software developers towards collaborating in a team using pair programming within agile software development;
- to assess the role of software development *subjective norms* towards using pair programming within agile software development by determining:
    - the managers' or team leaders' influence on software developers when using pair programming within agile software development,
    - peer influence on software developers within the organisation when software developers use pair programming within agile software development;
- to assess the role of *perceived behavioural control* in software development using pair programming within agile software development by determining:
    - resources/environmental influence on software developers when using pair programming within agile software development,
    - the software developers' perceived behavioural control's influence on their use of pair programming within agile software development;
- to assess software developers' *intention* towards using pair programming for current and future development within an agile industry

1.5 Overview of Methodology

This study will use case study research as a research strategy with the use of a mixed method approach using quantitative and qualitative data. Triangulation will be adopted to capitalise on the strengths of both qualitative and quantitative methods to provide accurate and concise conclusions. Data production will be underpinned by using the following tools: in-depth interviews (10

participants) and questionnaires (17 respondents). The text data will be analysed by using thematic analysis.

1.6 Limitations

The limitation of this study is the location as the researcher resides in the Durban region and there are few software development companies that practice pair programming with agile as a methodology. Only one company will be used for this study and the employees will be the participants; therefore the number of participants is limited. There are limits to the gathering of external and additional internal perspectives, such as from other software development companies and by using a wider range of participants.

1.7 Rationale

The topic of pair programming has been of great interest to the researcher for a number of years. It was first explored by the researcher in an educational setting in her Honours (Dhoodhanath, 2014) academic degree and the exploration was continued in an industry setting with the Masters study. The researcher is employed and functions within the department of agile software development but is not a software developer. The organisation where the researcher is currently employed does not make use of pair programming. However, the unique dynamics of pair programming remain of interest to the researcher.

Pair programming is discussed in the literature as an approach to successful software development (Williams, 2010). However, in a search for instances of the actual adoption of pair programming in software development, few were found. This is supported by Doyle et al (2014, p. 14) who states "we are 95% sure that no more than 25% of development teams have implemented pair programming". This discrepancy provides an interesting context for research, especially within an industry setting. Agile has also become a prominent topic in the Information Technology world; therefore the study focuses on agile specifically (Williams et al., 2000).

1.8 Summary of Chapters

This research paper is organised into eight chapters. The chapters begin with an introduction which will focus on the scope of the study. In addition to the problem statement, research questions, objectives and limitations of the study are presented in Chapter 1. This chapter advances to the literature review that will establish the background and underlying foundation that builds on the problem statement. The third chapter discusses the Theory of Planned Behaviour which is the conceptual framework that underpins the study. The chapter describes the logical association of the variables of the model, namely: attitude, subjective norms and perceived behavioural control. The fourth chapter discusses the research approach utilised in the study. The approach adopted is a mixed method case study and the chapter discusses the methodological factors such as research setting,

sample and sampling techniques. In addition, this chapter includes the process and administration undertaken for the development of the research instrument for the study. The method of analysis will also be discussed in this chapter, along with the ethical considerations. Thereafter, the following chapters will discuss the results/findings from the research, the fifth chapter analyses the quantitative, survey data and the sixth chapter will present the analysis of the qualitative data (interviews). The seventh chapter will answer the research questions in relation to both sets of data provided (survey and interviews) and will attempt to draw comparisons with what was discovered in the literature on pair programming. The final chapter discusses the conclusions of the study, as well as the limitations of the study and the recommendations for further study.

Chapter 2: Literature review

2.1 Introduction

The literature review was conducted by reviewing several past studies and online research libraries. Material was mainly sourced from the online research libraries of Google Scholar: http://scholar.google.co.za/ and the UKZN: http://researchspace.ukzn.ac.za/. This chapter will discuss the literature that supports this study. The first section will discuss agile software development as the overarching methodology that underpins the programming practise; thereafter the chapter will consider Extreme programming. Extreme programming is the software development methodology that focuses on using software development practises to the 'extreme' level. This embodies development processes that are more agile driven than traditional approaches that are perceived to be more restricted. Pair programming will be discussed subsequently along with the perceived benefits and factors that are noted as being admirable strengths. Hereafter, the challenges and difficulties experienced when using pair programming, which potentially contribute to the lack of its adoption will then be discussed. Finally, the conclusion will draw together the various discussions in this chapter.

2.2 Previous Literature Summary

The research on obtaining previous literature on the study involved reviewing online research libraries such as Google Scholar: http://scholar.google.co.za/ and the UKZN: http://researchspace.ukzn.ac.za/. Google Scholar is recognised as a legitimate source to use for searching for literature sources (van Aalst, 2010). The below table provides a summary of the Google Scholar search returns that were used for this study. Google Scholar was used as the main source. The searching method used was a funnel approach which began with the wider topic that relates to the problem statement i.e. agile software development, and narrowing the search to topics directly linked to pair programming in software industries and its benefits and challenges. The partial words used in the search string were used to obtain singular and plural relative topics for example: industr would include "industry" and "industries". The double quotation marks used within the search string are used to ensure two words are treated as a single unit in the search. There have been many studies that explored pair programming; but they were mainly conducted in an academic environment (Williams, 2010; Williams, Kessler, Cunningham, & Jeffries, 2000). Hence there were limited recent studies focusing on pair programming outside an academic context. The table 1 below provides the number of literature references related to pair programming in academic settings for the years from 2011 till 2014, this study began in 2015. On average there are 10 searches displayed per Google Scholar page

therefore the values reported as appearing on the first page are usually out of 10, and are sorted by relevance.

The results above show that research studies in pair programming and benefits have been mostly conducted in academia settings in 2011. The popularity of academia studies on pair programming, benefits and challenges have decreased slightly in 2012 which studies explored either both industry or academia and some only on industry. . In 2014 pair programming studies slightly decreased in academic settings, however in 2013 and 2014 the academic studies on benefits and challenges again increased towards academia settings. Therefore this study solely focuses on industry settings and is highlighted in light green in table 1 below to depict the years of the study duration to depict that this study contributes to the growing trend of using pair programming in industry settings.

Table 1: Previous literature on academic settings

Google scholar search	Total number of return per search string per year						
Number of studies relating to academic settings on page 1 of search	2011	2012	2013	2014	2015	2016	2017
"pair programming"	7	4	6	4			
"pair programming" benefits	8	3	4	7			
"pair programming" challenge	5	4	5	8			

Table 2: Previous literature stats summary

Google scholar search		Total number of return per search string per year								
Search String	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
agile software development	10300	12500	14200	15300	17600	18700	18600	21200	22100	22700
agile software industry	5830	6370	7360	8050	9150	10100	10000	13000	13100	13300
"pair programming"	788	801	900	954	1030	1110	1190	1190	1210	1250
"pair programming" software industry	436	439	514	587	619	665	652	693	774	754
"pair programming" agile software industry	327	322	392	441	473	495	490	496	576	541

The results in table 2 above indicate that research on software development has substantially increased for the last 12 years and has peaked since 2012 with a 220 increase in literature from 2011, and again from 2014 to 2015 by 260. This suggests that research on agile has rapidly increased in popularity during the subsequent years.

Past studies in pair programming have increased since 2008 however decreased slightly in 2013 but thereafter increased slightly in subsequent years till 2017. Previous studies relating specifically to pair programming in software industries have increased over the years but slightly decreased in 2014. However studies relating to pair programming in agile industries substantially increased over the last 12 years and jumped by 80 more studies from 2015 to 2016. This suggests that pair programming has been gradually gaining popularity within research over the recent 12 hence this study is relevant. Lastly past studies on benefits and challenges of pair programming also gradually increased over the recent 12 years which shows constant invested interest in these topics.

2.3 Agile Software Development

Agile has been used in the software development market since the 1990s. The initial intention of agile was to overcome known problems in the traditional/waterfall software development approach. Some of the issues include taking too long to deliver, and not being able to cope with a 21st century technological environment that is rapidly changing (Boehm, 2006). The aim of agile is to be lightweight, ensuring rapid delivery with continual improvements and a people-centric approach (Lindstrom & Jeffries, 2004). Agile software development is now more mainstream within software development industries (Doyle, Williams, & Cohn, 2014).

The need for agile came from the rapidly changing technological environments with countless needs, changes and desires. The traditional approach took longer to produce an outcome or accommodate a change, which made it difficult to cope in an ever-evolving technological environment. In addition, the traditional approach did not foster much interaction, which led to individuals producing output in isolation, which did not involve enough knowledge sharing, cross-skilling or productive teamwork (Balbes, 2014). Agile allows software to evolve, to consider and cater for changes and to break down outcomes into smaller increments so that delivery is more regular. By doing this, it also meets the recent and most updated needs of the users (Doyle et al., 2014). Agile in software development emphasises small software deliveries with fast paced cycles, with interacting parties constantly working together with high levels of communication. In addition, it results in adopting methods that are simple, which are easy to understand and follow, and which adapt to accommodate last minute changes, since the technological world is evolving rapidly (Abrahamsson, Salo, Ronkainen, & Warsta, 2017).

The agile manifesto documents the twelve agile principles (Lindstrom & Jeffries, 2004, p. 44):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in the development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to couple of months, with a preference for the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to, and within, a development team is face-to-face conversations.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity (the art of maximising the amount of working not done) is essential.
11. The best architectures, requirements, and designs emerge form self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behaviour accordingly.

The manifesto for Agile Software Development stresses the following (Lindstrom & Jeffries, 2004, p. 44):

- Individuals and interaction *over* process and tools
- Working software *over* comprehensive documentation
- Customer collaboration *over* contract negotiation
- Respond to change *over* following a plan

The above four statements summarise the value of agile: The word 'over' in each statement indicates that the statements on the left are preferred more than those on the right. The latter part of the statements refer to the 'behaviour' that is less important, but can be used if needed (Lindstrom & Jeffries, 2004). When developing software using agile, the importance of interaction between individuals is a priority; however, the adoption of processes and tools can be used to aid the primary focus of the final product. The use of documents can assist in defining rules/requirements for delivering working software. The use of contracts does assist in providing some conditions within

which parties can work; this becomes crucial when interacting with external clients. However the ongoing collaborations are what will develop the team to understand and deliver the appropriate final product (Doyle et al., 2014). Agile teams must be proactive rather than following a strict plan, since what was planned initially could have changed. Software development teams will need to respond to change continuously in order to deliver what the client wants at the point of delivery and not what was projected in a plan (Lindstrom & Jeffries, 2004). The interactive and iterative approach of Extreme Programming (XP) makes it the most pervasive approach in the agile methodology for software development (Lindstrom & Jeffries, 2004).

2.4 Explanation of Extreme Programming (XP)

XP embodies simplicity; with simple practises and sufficient feedback, the work is kept integrated and operational at all times (Lindstrom & Jeffries, 2004). XP allows an incremental planning approach, rather than planning everything upfront (Dudziak, 1999).

XP is a form of agile software development that conforms to the values and principles of agile. However in XP, development practises are exercised to extreme levels in the development process (Kaminsky, 2008). The XP values are communication, simplicity, feedback and courage (Konovalov & Misslinger, 2006). Communication is essential in software development and the information is disseminated to team members via a conversation, as opposed to the traditional approach of using comprehensive documentation. Through this rapid communication, developers develop a shared understanding between the developer (who needs to understand what the user wants to be developed) and the user (who shares details of what they want to be developed and solves queries during development) (Stankovic, Nikolic, Djordjevic, & Cao, 2013). The value of communication and simplicity is that they complement each other; for example, if the solution or system is simple then it makes communication easier, and more communication improves understanding, which leads to the simplest way possible to work. Feedback is also very important and can take various forms: For example, feedback from the developer develops interaction; which ensures that the right thing is being built and designed to mitigate risks. Feedback from the developers can also provide an estimation of how long it takes to produce a desired deliverable. In all instances, feedback must be continuous and given as soon as possible so that risks are mitigated as early as possible. Courage in XP enables developers to refactor their code, by reviewing current systems when making modifications, so that when the system is updated in the future, modifications can be implemented easily (Konovalov & Misslinger, 2006).

In software development, XP has a set of core practices: plan, design, code and test. Planning in XP is when the stakeholders e.g. the client who wants software to be developed, will list the desired features that they want to be delivered (Dudziak, 1999). The deliverables must be clearly defined, including details that describe what is required. This then allows the developers to estimate how much

effort it will require to produce in a given time, which is known as the 'iteration' (Dudziak, 1999). This details how the planned product will be produced and no extra functionality is allowed; only what has been planned for will be considered. However at times there are tough or difficult questions during software development that cannot be answered immediately due to the lack of information at that stage. In these situations XP suggests the use of 'spike', which is a design prototype that explores potential solutions for only one problem at a time (Konovalov & Misslinger, 2006). A particular portion is implemented and evaluated which is meant to help lower the risk when the actual implementation begins. In addition, XP encourages 'refactoring' in the design, which is an iterative refinement to the design. The aim of this is to minimise the chances of bugs when coding and makes the design easy to understand and modify (Konovalov & Misslinger, 2006).

In XP it is recommended that when coding the software the developers should be on the client's site. This facilitates frequent communication and interaction with the actual user of the software being produced (Dudziak, 1999). The coding standards of XP allow any developer to modify the code and ensure that the communal code is maintained and is consistent for the entire team to read or modify. There must also be continual integration which is generally done by 'unit tests'. Unit tests are a series of tests or validations that will execute the desired product (Dudziak, 1999). Once the code is completed, then it is immediately tested in a unit. The developers should then continually release their code in the code repository; by integrating small amounts it helps integrate the entire system (Konovalov & Misslinger, 2006). Lastly, in testing, acceptance tests (also known as 'customer tests') are performed.

2.5 Pair programming

This section will describe the dynamics of pair programming and discuss the perceived benefits and challenges associated with the use of pair programming.

2.5.1 Description of Pair Programming

Pair programming is a process that forms part of XP within the agile methodology. This programming approach involves the collaborative efforts of two software developers constantly sharing ideas and working together as one, on one task (Williams, 2010). Since pair programming is an agile method, the inherent dynamics of agile are instilled within this practise.

Pair programming is highly encouraged as a core XP practise. While pair programming involves two developers sitting side by side, using only one computer, working on one task (Williams, 2010), each developer works at a different level of abstraction, so that one developer will be the 'driver' and the other will be a 'navigator'. The role of the driver is to write the code, while the role of the navigator is to review the code being written; searching for tactical errors, refactoring or thinking of strategic/architectural issues that may arise (Cockburn & Williams, 2000). These two roles are always

present when coding. Coding using pair programming, however, requires that the two roles are swapped at regular intervals. This means that a developer can only fulfil one role at a time, but will have the opportunity to practise both driver and navigator roles when developing the code. The roles are switched to avoid animosity and to maintain the integrity of the team. In addition, since each developer fulfils both roles, it prevents either developer playing a dominant role as they are both active participants in the development and contribute to a successful outcome with collective ownership (Williams, 2010). The agile manifesto strongly promotes interaction over documentation, and this aligns with pair programming since it is based on interaction between the software developers. This interaction is not just about discussing the task on hand, but it also enables the individuals to understand the reasoning behind the process, and this facilitates a sharing of knowledge (Doyle et al., 2014). The adoption of pair programming has been relatively low, possibly due to a perceived imbalance between the benefits realised by the use of pair programming and the negative factors associated with its use (Doyle et al., 2014).

2.5.2 Perceived Benefits of Pair Programming

The agile manifesto strongly promotes interaction over documentation, and this aligns with pair programming since it is based on interaction between the software developers. This interaction is not just about discussing the task on hand, but it also enables the individuals to understand the reasoning behind the process, and this facilitates a sharing of knowledge (Doyle et al., 2014). This programming approach involves the collaborative efforts of two software developers constantly sharing ideas and working together as one, on one task (Williams, 2014; Williams, 2006; Dillenbourg, 1999). The effort of collaboration and participation mediates the process of sharing knowledge which inherently assist to boost self-esteem as the individual becomes more knowledgeable may feel more empowered. The purpose of sharing knowledge is commonly used within industries to upskill a junior or less experienced developer.

A recent study conducted by Tsyganok (2016) shares the interaction experiences of a senior and junior developer based in an online retail store in US, when using pair programming. Tsyganok (2016) discusses the pairing relationship between a beginner, who is a graduate straight out of university, and a senior developer. The initial purpose of this pairing was for the graduate to leverage off the senior developer as a ‘training tool’. The training required to upskill the beginner helps build his confidence levels. The interaction allowed the beginner to interact freely as a relationship was built with his partner. The guidance of a more experienced developer helped the beginner understand the way of working in the company. By using this approach in pair programming there is also constant communication and reviewing of code. This constant interaction allows continual comparison and exchange of ideas between the two developers. This helps the developers have a clearer understanding of the problems or possible solutions (Tsyganok, 2016).

Pair programming is also noted to be the most efficient code reviewing technique that generally delivers a higher quality of work (Fu et al., 2017). Due to the constant communication and sharing of knowledge when practising pair programming, the developers tend to have a good understanding of the code being written. The presence of the two roles, that of driver and navigator, provides an opportunity to master the ability to code and the ability to review (Schmidt, Kude, Heinzl, & Mithas, 2014). A Norwegian study conducted by Fu et al. (2017) used pair programming as a technique to review code. The results of the study show that in the early stages, more defects in the code were discovered and fewer tests were needed later. The code quality was greatly improved due to the navigator role being mandatory. This enabled detailed checking and improvement of the quality. Besides producing code of good quality, this also allowed developers to explain and actively engage and thus to develop a common understanding of the task; including increased sharing of knowledge (Fu et al., 2017).

This contributes to the sharing of knowledge between developers which helps them leverage off each other's skills and abilities. The strengths and weaknesses of both developers are accommodated, which helps to improve productivity (Balijepally, Mahapatra, Nerur, & Price, 2009). The constant engagement of both individuals is also a process that aids collective ownership of a successful outcome.

Despite the range of benefits realised with the use of pair programming, previous studies mention its disadvantages, which could be the reason for the low adoption rate of pair programming within industry.

2.5.3 Challenges with Pair Programming

The above-mentioned benefits would seem to be favourable to software development. However, in reality, the adoption of pair programming in industrial settings is very low (Doyle et al., 2014). Research conducted by Williams (2010) noted that one of the reasons for the low adoption of pair programming is that developers at software industries feel that the dynamic of pair programming is unfamiliar and varies significantly from the traditional way of operating; which is individually programming at individual terminals (Cockburn & Williams, 2000; Williams, 2010). It took the developers several days to become familiar with the dynamics of pair programming and it took longer for them to conform to the practice. Software developers became very uneasy when required to work in pairs for the entire working day (Kim, Barnes, Chan, & Keith, 2010). The time taken for developers to transition from traditional/solo programming to pair programming initially increased the time taken to complete a task, since individuals spent most of the time familiarising themselves with the dynamics of the new technique, before putting it into practice (Williams, 2010). This suggests that the norms of the company will impact the use of pair programming. For example, if the company only

values fast results instead of spending time to upskill individuals to conform with the practice, then this prevents the continuing use of pair programming (Tolfo & Wazlawick, 2008).

The constant conversation between software developers engaged in pair programming is seen as tiring due to the lengthy hours of being involved in discussions. This can become a possible cause of conflict if personality clashes arise due to contradictory personalities; for example during pair programming an extrovert may overpower an introvert because of their personality, which may lead to them being seen as bossy, belittled etc. (Kim et al., 2010).

Vanhanen (2005) has noted that it takes the right combination of individuals in a pair to be productive when using pair programming. Williams (2006) noted that the definition of compatibility is when the attributes of one individual are deemed favourable to another individual. Williams (2006) defined six attributes that were used to determine compatibility: personality, learning style, skill levels, programming self-esteem, work ethic and time management. The experiment described in that study was conducted in an academic setting in US. The results of that paper indicate that individuals prefer to pair with someone whom they perceive to have a similar technical competence to themselves. It also concluded that pairing Myers-Briggs sensor-intuition personality types yielded very compatible pairs (Williams, 2006). 'Sensors' are defined as individuals who prefer to obtain information through experience and pay attention to details. An individual with an 'intuition personality' prefers obtaining information through abstract concepts or innovative thoughts (Williams, 2006).

Interestingly, time management had no significance, indicating that time management may not be a factor that affects pair compatibility. Work ethic in Williams's (2006, p.7) study was defined as "when another individual who has similar ambitions for success", by which individuals with similar worth ethics are more likely to work well together. The results indicate that individuals with similar work ethics are more compatible (Williams, 2006). It was also noted that individuals with a low self-esteem liked using pair programming more than those with a higher self-esteem. Since pair programming is also used as a tool to upskill, those with low self-esteem are generally reluctant to approach a task individually (Williams, 2006). Those pairs that were incompatible were characterised by a lot of differences that sometimes caused a barrier in communication and they found it difficult to express their views, limiting mutual understanding.

Incompatible pairs take longer to get consensus and also waste time on discussions. It is also possible that, with incompatible pairs, discussions become arguments which result in a bad relationship and no consensus (Begel & Nagappan, 2008). Conflict in the pairing often resulted in work of an average quality. The self-esteem of individuals in an incompatible programming pair suffered, so that they became tired and hesitant (Williams, 2006). Partners in pairing should be open to thoughts and ideas and actively listen and participate but not dominate the situation. Since communication is key in pair

programming, the more comfortable an individual is when conversing with another person, the more comfortable they will be to receive and give negative comments or criticism (Begel & Nagappan, 2008).

Williams (2010) mentions that the co-ordination of tasks and hours spent becomes a challenge, since this needs to be reassessed often. Pair pressure can be positive or negative: if the team works well together they can inherently transfer their good or bad habits to each other, which will affect the end result. Pair negotiation and brainstorming are important components of pair programming that contribute a lot to the success of pair programming (Williams, 2010). If developers are not able to negotiate effectively or brainstorm, then solutions will not be devised efficiently and this may cause frustration and decrease productivity (Williams, 2010). Williams (2010) indicates that this can be resolved by ensuring the partnership shares the same goals, so each considers the other person's suggestions and the pair jointly determines the best approach.

Pair programming has been noted to be more suitable for centrally located teams than geographically dispersed teams. This is because the design of pair programming makes it more maintainable when developers are sitting in the same location, and as close as possible (Fu et al., 2017). However, there are some disadvantages to pair programming when set up in a 'classic pair programming fashion' (Fu et al., 2017, p. 2). Fu et al. (2017) identified some of the issues: when the drivers code aloud, then the navigators may be discouraged and not voice themselves. Eventually the process reverts to solitary programming, with the drivers checking their own code due to the silence from the navigators. In addition, pair programming can be a tiring practice as the continual interactions can be mentally and physically draining (Fu et al., 2017).

One measure to mitigate the animosity between roles was to allow a day to lapse before swapping roles. This meant the developers spent the same amount of time performing responsibilities of both implementing and testing. Also, being able to work at his or her own terminal guaranteed the developer's autonomy, who then felt free to try and test an approach before having to convince the other developer. When one developer focused on one responsibility for an entire day, it allowed him or her to gain a stronger understanding. Williams (2010) indicated that a suitable time for pairing in pair programming is between 1.5 and 4 hours a work day. The handover of responsibility on the next day gave the developers a chance to start afresh on the responsibility. The developers felt it is necessary to understand the work already completed thoroughly in order to continue with the next phase of the work (Fu et al., 2017).

2.6 Conclusion

Agile methods have gained dominance within software development industries. The favourable principles of the methodology result in it being deemed to be the most successful approach to software development. Pair programming, belonging to agile, also results in some of these benefits; for example, the interaction improves sharing of knowledge and produces higher quality of code. The pair programming approach also promotes collaboration that assists in upskilling inexperienced staff. Since the work is completed in a team this boosts confidence levels, leaving those who may have been feeling demotivated, feeling more empowered. However, the adoption of pair programming has been very low within industries. The challenges of pair programming make software developers withdraw from the process as the dynamics of pair programming vary significantly from the traditional approach. For instance, software developers find working in pairs for lengthy periods drains them mentally and makes it difficult to cope. They also find themselves taking several days to become familiar with the dynamics which slows down their progress and makes them frustrated.

It appears that the challenges experienced when using pair programming discourage the software developers from continuing with its use. The negatives experienced when using pair programming influence attitudes towards the practice. In addition, in a technological environment with ever-changing needs that require attention, the company's focus may be more on delivery and output. This also tends to make software developers focus more on output and delivery rather than learning to conform to the dynamics of pair programming. It is crucial to have a work environment and resources available which are conducive to the use of pair programming as it differs vastly from the traditional approach; for example, having desks set up for two individuals to work on one computer would facilitate the use of pair programming and reduce the negativity felt towards the practice. The more negative the attitude is towards pair programming (including company culture or even demotivated or negative colleagues), and not having the correct resources to make implementation easier, the less pair programming will be used.

Chapter 3: Conceptual Framework: Theory of Planned Behaviour (TPB)

3.1 Background and Context of Conceptual Framework

The Theory of Planned Behaviour (TPB) originated in the mid 1980's as an extension to the Theory of Reasoned Action (TRA). The TRA was introduced by Fishbein in 1967 as a model to study human behaviour and is made up of two constructs: namely, attitudes and subjective norms. The influence of these factors motivates an individual to perform a specific behaviour (Icek, 2006). Ajzen (1991) notes that humans behave in a rational manner by using the information immediately available to them to act on an intention and perform a specific behaviour (Icek, 1991). Therefore, the TRA is used to define how an individual intends to behave by considering the personal aspects of attitudes and the perceived social pressure or subjective norms (Icek, 2006). However, both TPB and TRA have a central focus on an individual's 'intention'. The intention captures how hard people will try to perform behaviour. For instance, if the intention is strong then an individual is more willing to engage with a behaviour (Icek, 2015).

When TRA was applied, many shortcomings were noted; one of the biggest being the inability to control human behaviour. Human behaviour depends on the degree of individual control of attitude and behaviour; for instance, an individual can feel they have more power or little power over their behaviour (Icek, 1991). Ajzen and Fishbein continued to refine, develop and test ways to improve the predictability of the model; and a third element was added to the original model, which is perceived behavioural control. This factor is used to assess the extent to which an individual is able to exercise control over performing a specific action. For instance, the more effort an individual invests and the greater their control over the external factors that may influence the behaviour, then the greater the likelihood that the individual will achieve the outcome. Hence, perceived behavioural control motivates the behaviour (Icek, 2015).

The aim of TPB is to predict and understand the factors that influence an individual's behaviour that is not "under volitional control" (Armitage, 2001, p. 472). The main factor remains the behavioural intention, which indicates that individuals consider the implications of their actions before they engage with behaviour. The extent and control an individual has over the implications will determine how hard an individual is willing to try to perform the behaviour (Icek, 2015). The Theory of Planned Behaviour consists of three components that contribute to the behavioural intent: attitude, subjective norms and perceived behavioural control. The factor 'attitude' (positive or negative attitude) indicates that if individuals perceive that, by performing a specific behaviour, the outcome will be positive, they will have a positive attitude towards performing the behaviour. The factor 'subjective norm'

(motivating or demotivating) indicates that if individuals feel that people close to them see the outcome of the behaviour as being positive, then the individual is motivated to perform the behaviour. The last factor is 'perceived behavioural control' (easy or difficult to control). The addition of this construct is the main difference between TRA and TPB. Perceived behavioural control is a factor which indicates that if the individual believes he or she has a strong or weak control on the existing factors that facilitate the behaviour, then this will then influence the control of the behaviour. If a person feels that it is difficult to perform the behaviour, in conjunction with the perception of not achieving a successful outcome, then this will demotivate the individual and control the behaviour to be, or not to be, performed (Icek, 2015).

Previous studies had noted that the weakness of TPB is that the model only considers the psychological aspects of behaviour. Physical elements, such as body language, tone, voice etc. are not considered. TPB may thus not provide a holistic context to the understanding of how the individual feels or reacts (Icek, 2006). In addition, individual habits limit this theory, as habits create a sense of comfort and ease for individuals. Therefore, if an individual is forced to behave contrary to his/her habits, it creates discomfort and reluctance to perform the behaviour. This theory lacks a factor that takes this into account and hence falls short in the provision of a holistic view of the individual's behaviour (Withagen, 2007). However, the application of TPB to a particular area provides a lot of information that is very useful in understanding behaviour and in targeting strategies of how and where behaviour changes. For instance, intention, attitude, subjective norm and perceived behavioural control each reveal a different aspect of the behaviour that can also each target strategies to change it. The underlying focus of TPB is 'intention' and this facilitates an understanding of the unique factors that induce an individual to engage in a behaviour or prompt a different course of action. In addition, TPB focuses on the intent of behaviour as well as the actual behaviour, so both factors are evaluated. This eliminates bias and allows results to be compared with the intent and actual actions (Icek, 2006).

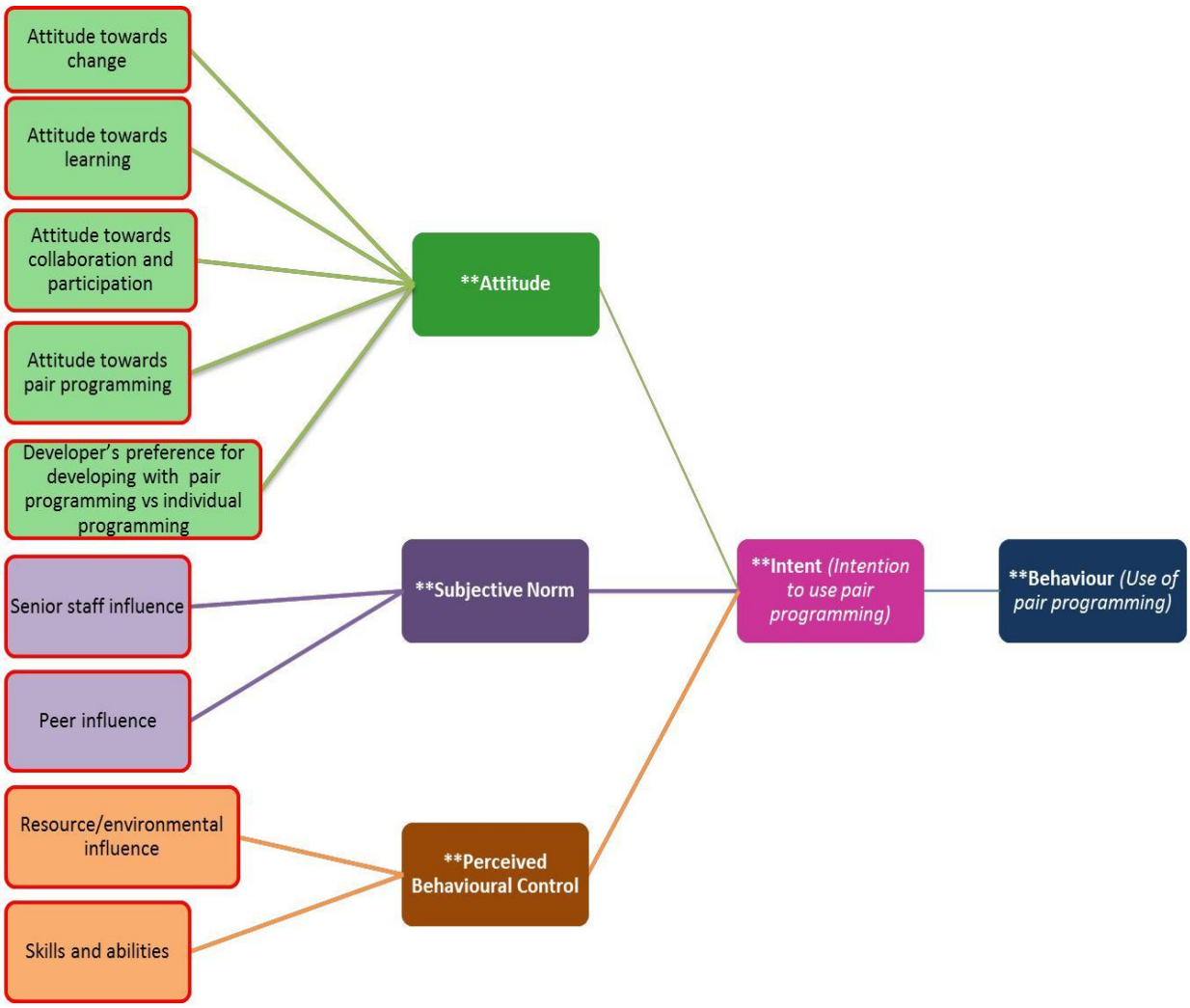
Asaria et al. (2014) analysed the TPB and extended the theory by adding realistic elements to improve its predictive power and its effectiveness in aligning realistic measurement with behaviour. Therefore, this study will adopt a decomposed model that has been developed by using Ajzen's TBP factors as a conceptual framework and Asaria et al.'s study to devise the realistic measurements for the sub-factors of TPB: attitudes, subjective norms and perceived behavioural control.

3.2 Application of Conceptual Framework

Due to the limitations of TBP in depicting realistic factors (attitude to the change, subjective norms of senior staff), this study has extended the standard constructs of the model to add in those specified within the decomposed model based on Asaria et al.'s work to add the realistic elements. The symbol ** within the blocks in the diagram below indicates the standard constructs of Ajzen's TBP; whereas the blocks highlighted in red are the additional constructs from the study conducted by Asaria et al.

(2014). The section below the diagrammatic representation is a detailed explanation of the variables and will also indicate the studies from which the variables were sourced. The explanation will portray the conceptual framework of the Theory of Planned behaviour by Ajzen (1991) and how it's aligned to the additional variables developed by Asaria et al. (2014).

Figure 1: Decomposed Theory of Planned Behaviour with elements from Ajzen (2006) and Asaria et al. (2014):



3.2.1 Attitude

This factor is determined by assessing whether the attitudes of software developers are positive or negative (Icek, 2006). Ajzen notes that individual behavioural belief provides insight into the likely outcome of behaviour. Behaviours are the desired outcomes that are actioned after considering more favoured and less favourable attributes (Icek, 2006). The following variables will measure the factor 'attitude': in essence, the intention of software developers to use pair programming, compared to the behaviour of the actual use of pair programming:

- **Change:** Developers' reaction to change can have an impact on their attitude towards a specific task or approach to a task. An agile-oriented industry is often susceptible to change; this can make developers feel motivated (i.e. have a positive influence) or challenged (i.e. have a negative influence) (Asaria et al., 2014).
- **Learning:** Since change is constant, attitudes towards learning, if favourable, can improve knowledge, skills and abilities. This makes the attitude of the developers towards a specific task more positive, which makes them more willing to perform the task (Asaria et al., 2014).
- **Collaboration and participation:** The agile principle strongly emphasises collaboration which enhances teamwork and allows a holistic consensus of what needs to be done. If there are positive attitudes towards collaboration and participation, this builds stronger team relationships and improves self-morale. The positive attitude will impact the behaviour of team members by making them more willing to collaborate and participate. The more developers collaborate and participate, the more productivity improves (Asaria et al., 2014).
- **Attitude towards pair programming:** This construct discusses the specific positive/negative attitude adopted when engaging with pair programming. This will help ascertain the feelings that either motivate/demotivate the practice of pair programming, currently or in the future. For example, the more positive the attitude to engaging with pair programming, the more motivated developers are to perform pair programming (Asaria et al., 2014).
- **Developer's preference for developing with pair programming versus individual programming:** The traditional approach of individual programming prevails in software development. The dynamics of pair programming, which is so different to individual programming, creates a discomfort, that makes developers reluctant to use it (Williams, 2010). Therefore, this construct is designed to determine whether the developer has a positive or negative preference for using pair programming versus individual programming.

3.2.2 Subjective norm

The second major factor of the TBP that influences intention is 'subjective norm'. This factor is determined by assessing how significant others (people that the individual relates to) motivate or demotivate software developers in terms of the expectation they create of what is considered 'normal' behaviour. The social pressures received from these individuals creates a belief within software developers in relation to what people in their work environment believe about them practising pair programming (Tao, 2008). The following variables will measure the factor 'subjective norm' to determine the motivating or demotivating factors that influence the intention of software developers in the use of pair programming, relative to the behaviour of the actual use of pair programming:

- **Managers' or team leaders' influence:** An individual at a higher level of leadership either creates barriers or enablers for software developers to perform pair programming (Asaria et al., 2014).

- **Peer influence:** This factor refers to the views, opinions and motivations of work colleagues or fellow team members that impact a person's belief about a certain behaviour (Asaria et al., 2014).

In both of the above cases, if the manager/team leader or peer has a positive attitude towards pair programming then the subjective norm from these individuals has a positive influence on the behaviour. The positive influence increases the motivation to engage with pair programming.

3.2.3 Perceived behavioural control

The third and final major factor of TBP that influences the intention is 'perceived behavioural control'. This is determined by assessing the ease or difficulty associated with performing the behaviour, based on the control the software developers perceive they have over the situation. The control belief includes factors that will facilitate or inhibit the performance of the behaviour which can be external (such as physical office layout) or internal (such as skills/abilities). This factor will be measured by the control beliefs a software developer has on using pair programming and the perceived importance of the use of pair programming within an agile industry (Icek, 2006). The following variables will measure the factor 'perceived behavioural control' with regards to the intention of software developers' use of pair programming in relation to the behaviour of the actual use of pair programming:

- **Physical setting and resource availability factors:** The available resources, such as hardware, the office layout or the positioning of desks will impact the use of pair programming. The more readily available resources are to accommodate the needs of pair programming, then the more positive the attitude of the developers will be to use pair programming. If the resources are difficult to attain, or there is a lack of resources, then the developers will be reluctant to engage with pair programming (Asaria et al., 2014).
- **Skills and abilities:** The knowledge and skills they have will determine the confidence level of software developers. If their skills and abilities are well aligned to pair programming, then the developers will find it easier to use pair programming. Therefore, their skills and abilities will impact the ease or difficulty with which software developers use pair programming (Asaria et al., 2014).

3.3 Conclusion

This study makes use of the three main constructs of TBP and the additional measures developed by Asaria et al. (2014) to determine the various perceptions of software developers when using pair programming in agile software development industries. This chapter provides a clear insight into the factors to be considered within these constructs as a platform for the conceptual framework. The aim is to conceptualise these factors in the industry setting and then determine the perceptions of software developers when engaging with pair programming. The next chapter will consider how the study is structured to allow the collection of the appropriate data for each factor.

Chapter 4: Research Methodology

4.1 Introduction

The literature pertinent to the problem statement is reviewed in Chapter 2, which discusses the inherent benefits of, and issues derived from, the use of pair programming. The literature mainly focuses on experience in industry to assist in understanding the use of pair programming, notwithstanding the relatively low adoption of pair programming. Kim (2010) found that the dynamics of pair programming are so different from the traditional approach that individuals are dissuaded from using it. Also, having two resources working on one task is perceived as a waste of resources (Kim et al., 2010). Chapter 3 focuses on the conceptual framework adopted in this study. The chosen framework is the Theory of Planned Behaviour, which consists of three main constructs: attitudes, subjective norms and perceived behavioural control. The constructs of this model will be the basis of the measurements that will be used in the research instruments. This chapter will discuss the details of the methods that will be is used to obtain data. This chapter will discuss the source of the research data, statistical analysis and ethical considerations.

In order to gain a better understanding of the factors that influence the software developer to use pair programming, a mixed method approach has been adopted, using questionnaires and interviews as research instruments to collect data that will be analysed to address the research questions. The advantage of a mixed method is that it "combines the strengths of qualitative and quantitative methods and compensates at the same time for the weakness of each method" (Tashakkori & Teddlie, 2010, p. 290). For example, quantitative methods reveal relationships and conceptualise variables; while qualitative methods give detailed insights and context to the study.

4.2 Problem Statement

The study conducted by Doyle et al. (2014) indicates that the adoption rate of agile within software development industries is continually growing and the outcomes derived from using agile are positive and productive (Asaria et al., 2014). However, Doyle (2014) also indicates that one of the least used agile practices in software development industries is pair programming, especially by new teams. This creates a discrepancy between the adoption profiles for agile and pair programming. It has also been noted that there is a dearth of studies that explore the motives for, and impacts of, using pair programming within an agile software development methodology (Doyle et al., 2014). Since there is a lack of understanding of what happen in software development industries, this study is important to facilitate understanding and evaluation of the factors that create barriers to the adoption of pair programming within an agile software methodology.

In light of the problem statement identified in Chapter 1 (section 1.2 Problem statement), and taking into consideration the literature review based on the use of pair programming within industry, and the reasons that influence its adoption (mentioned in Chapter 2, section 2.4.2 Perceived Benefits of Pair Programming), the true nature of the problem highlighted in this study is identified. Past research indicates that pair programming and the lack of adoption of this technique is an ongoing issue faced within industries. However, the complexity of this issue has not been fully covered. Therefore, this study is justified by the problem statement which is: "Factors influencing software developers' use of pair programming in an agile software development methodology environment". There have been many studies that explored pair programming; but they were mainly conducted in an academic environment (Williams, 2010; Williams et al., 2000). There were few studies that explored pair programming within industry, and no study has been conducted in an agile software development setting. Hence the questions posed in Chapter 1 remain as stated.

4.3 Research Design

A descriptive research design has been adopted for this study, due to its effectiveness in bringing the researcher to an understanding of the problem at hand; and its flexible approach to learning and understanding the problem in a real life context. As human behaviour is so complex and potentially irrational, a descriptive study provides a structured approach to making sense of the phenomenon. Descriptive research is flexible in the acquisition of information through quantitative and/or qualitative approaches (Sekaran & Bougie, 2013). This allows the researcher to focus on understanding the relationship between variables that influence the use of pair programming (Sekaran & Bougie, 2013).

A case study research strategy is used with mixed method approach, which is a combination of quantitative and qualitative approaches, has been used for this research. Due to there being limited empirical evidence of the problem statement. The company chosen for the case study followed a criteria mention below in section 4.4. The purpose of the case study approach is to collect information about the problem statement from a company so that a clear picture is obtained with a real life situation. By using mixed methods the case study is able evaluate the problem statement from various angels and using multiple methods (Yin, 2017). The study makes mention of mixed methods as this is the underlying research strategy of the study and is used to achieve triangulation. Therefore the decision to use mixed methods as a strategy is to combine the qualitative and quantitative methods to capitalise on the combined strengths and compensate for the weaknesses of each method (Tashakkori & Teddlie, 2010). For instance quantitative methods are known for its strength of conceptualizing factors and relationships whereas qualitative methods are known for providing context, meaning and in depth understanding (Tashakkori & Teddlie, 2010).A quantitative approach was used in the survey

using questionnaires as a research instrument and the qualitative approach was based on interviews. This study first uses surveys to garner a broad idea of the perceptions within the organisation. The surveys help to ascertain the important factors that can be further researched by using interviews. The intention of using the interviews is to probe for refined information.

4.4 Research Site

4.1.1. Selection of the site

The process used to select the site included three criteria that were set by the researcher. The criteria were:

- The company should be located in the same location/region as the researcher, so that it would be convenient for the researcher to be physically at the company's site. The travelling expenses would be less costly as there would be no need for accommodation or flights, and minimal electronic interaction costs would be involved. Besides the convenience, the 'same region criterion' was adopted to allow most of the data collection to be personally administered by the researcher. The benefits of personal administration included the immediate resolution of queries and misunderstandings and ensured a higher response rate.
- The company should be active in the software development industry. This is necessary as the participants had to include software developers. The use of software developers directly relates to the topic on hand.
- The company should currently be using agile and pair programming. This is necessary as the research questionnaires require respondents directly connected with the topics, who are able to understand and relate to the questions. This should lead to the collection of sound data.

The researcher used the above mentioned criteria and shortlisted three companies. However, the last criteria, 'currently using pair programming' were one of the most difficult to satisfy. Only one company of the three was chosen. Two of the companies were prepared to participate in the study; but only one company actively engaged in pair programming. One of the companies was considering the implementation of pair programming; but the company using both agile and pair programming was chosen as it was most aligned to the study.

4.4.2 Discussion of the research site

The population used in this study are the software developers from a specific company based in the Durban region of KwaZulu-Natal. Durban was chosen as the location as the researcher resides in the same region and therefore travelling to the company is convenient and economical. The company has chosen not to have its name specified in the research.

This company is a software development enterprise that services both local and international markets, adding value to its clients by providing development services, training and knowledge for workers to improve their working abilities. This company also met the purposive criteria as it is a software development house using agile methodologies that allows its development leads (people who take the lead on a specific development project) the freedom to choose a particular programming practice. An additional advantage of this company is that it currently practices pair programming and other programming methods.

4.5 Sampling and Sampling Technique

The company comprises five administrative staff and one project manager who are not developers and 21 software developers. The company employs software developers at various levels: senior developers, team leaders and developers. The software developers work at the company office or as a consultant, onsite at client's premises. Only 17 software developers responded in the survey. Some individuals were excluded at the request of the company as they were core staff. Others were not available to participate as they were not at work. There were also some individuals who chose not to participate in the study. Only ten participants were interviewed as that was the maximum number permitted by the company. Since the study utilises two interview schedules – one for senior staff and another for developers – the researcher requested five individuals per group. The participants for the senior staff interviews were senior developers, team leaders or managers. The participants for the developer interviews were junior or intermediate developers. Therefore, the company had allocated suitable individuals for the interviews according to their availability. Only a total of 17 software developers participated in the study, out of a possible 21 software developers.

The targeted sample is influenced by the selection of the company as this is part of the convenience and purposive sampling technique that is adopted in this study. The developers belonging to the company are extensively involved in software development and team work. They work in teams and therefore as a team decide the programming approach they adopt. The approach generally varies with each project and the approach decided upon is understood and agreed upon by the team. The duration of the projects is not fixed and varies.

The study makes use of a census approach for the surveys as the population size is small and the aim was to involve all individuals who met the selection criteria. Therefore the study could not make use of other sampling methods, such as random sampling, as there was limited access to companies and individuals.

4.6 Data Collection

4.6.1 Rationale for selecting research instruments

The data assessment instruments used in this study are surveys and interviews. These instruments allow for sufficient data to be collected through closed-ended questionnaires and structured interviews. The process of selecting an appropriate approach has been rigorous and the necessary measures have been followed to ensure the most appropriate approach is adopted (Tashakkori & Teddlie, 2010). In addition, statistical tests and other measures are used to substantiate the results. The closed-ended questionnaire was chosen to ensure that participants' responses were more reliable and accurate (Sekaran & Bougie, 2013). Structured in-depth interviews were conducted with senior level developers (team leads, managers) and some developers (junior, intermediate developers). Surveys were conducted by administering questionnaires to all developers at different levels of expertise and who use different programming methodologies. This study first used surveys to establish a broad idea of the perceptions within the organisation. The surveys helped ascertain the important factors to be further researched by using interviews. Interviews were used to probe for refined information.

Questionnaires

"A questionnaire is a pre-formulated written set of questions to which respondents record their answers, usually within rather closely defined alternatives" (Sekaran & Bougie, 2013, p. 147). The questionnaire is an effective research instrument due to its ability to collect structured perceptions, which can be represented as numeric data. Questionnaires allow for information to be easily collected and involve minimal administration. They are also inexpensive and easy to interpret (Cohen, Manion & Morrison, 2007).

Personally administered questionnaires are used to provide context and initial information which is used during interviews to help establish a rapport and improve the quality of the discussion, which improves the quality of data collected (Sekaran & Bougie, 2013). Any queries can be resolved immediately, facilitating a high response rate (Sekaran & Bougie, 2013).

The questions differ between the various levels of developers within the company (team leaders, juniors, and senior developers). The categorisation by level is adopted to facilitate the comparison and understanding of the role of knowledge, attitudes and behaviour in the developers' use of pair programming (Sekaran & Bougie, 2013).

Interviews

"The use of interviews in research marks a move away from seeing human subjects as simply manipulable and data as somehow external to individuals, and towards regarding knowledge as generated between humans, often through conversations" (Cohen et al., 2007)

An interview allows the researcher and respondent to exchange their views personally. The human interaction allows the researcher to explore in more detail the respondent's experiences and views on the research topic (Cohen et al., 2007). The benefit of using interviews is that respondents are able to freely share more detail about their views or experiences and clarification can be provided immediately. Also, the respondent's body language, voice tone and facial expressions are easier to read in an interview, which may help provide more details about responses (Sekaran & Bougie, 2013). In order for the objectives of this research study to be realised structured, personally administered interviews have been used.

Structured interviews allow eliciting more in-depth information from the interviewee. There were predetermined questions that the participants were required to answer. In this study, the researcher interviewed employees from different work levels in the working environment, such as managers, team leads and developers. The type of questions asked varied according to the level of the employees. The biggest advantage to using this type of interview is that preliminary or unforeseen issues can surface, allowing the researcher to determine the factors that require further investigation (Sekaran & Bougie, 2013).

Personal administration of the interviews allows the researcher to interpret responses more holistically. For instance, non-verbal clues can be observed and the researchers can easily establish a rapport with the developers. Also, any clarification can be sought, or misunderstandings cleared up immediately; allowing rich data to be obtained (Sekaran & Bougie, 2013).

4.6.2 Process of developing the questionnaire and interviews

In order to develop the statements included in the questionnaire and the questions asked in the interviews, an alignment matrix was constructed. Below is a snap shot of the alignment matrix, which has been provided to assist the detailed explanation of the alignment matrix. The full alignment matrix can be located in Appendix E, showing the alignment of research questions, variables within the research questions and the actual measurement variables.

Figure 2: Snap shot of alignment matrix for questionnaire

Note: The actual question within the alignment matrix only refers to the question number. The actual questions of the survey can be found in Appendix B. The actual questions of the interviews can be found in Appendices C and D.

Main Research Question	Variables in Research Question	Measurement of Variable of Survey	Measurement of Variable of Senior Staff Interview	Measurement of Variable of Developer Interview	Survey Question no.	Senior Staff InterviewQuestion No.	Developer Interview Question No.
Profile					1		
					2		
					3		
					4		
					5		
1. Use of pair programming	Use	How **long** have you been **using pair programming**?	Pair programming ***experience***		A.1	A.3	A.1
					A.2	A.4	A.2
				Factors relating to working with a ***partner***			A.6

The alignment matrix was structured by using the research questions and the constructs of the Theory of Planned Behavior. The first column (Main Research Question) of the matrix highlights the research questions posed in this study (as mentioned in section 1.3). The research questions each focus on a different concept, which avoids redundancy. The second column (Variables in Research Question) is more refined and relates to the constructs of the conceptual framework of this study which is the Theory of Planned Behavior. The constructs are attitude, subjective norms, perceived behavioral control and behavioral intention. The aim of the second column is to highlight each variable that the research question will validate or measure. The third, fourth and fifth columns are more detailed, breaking down the variables in the second column to relate further to the actual measurement for the conceptual framework constructs. The columns 'Survey Question No (Number)', 'Senior Staff Question No', 'Developers Interview No' are the actual question numbers used in the interview and questionnaire. The detailed questions are articulated in the survey and interview schedules. Appendix B contains the survey sheet with the actual questions asked. Appendices C (Interview Schedule for Senior Staff) and D (Interview Schedule for Developers) contain the interview schedules with the actual questions asked.

4.6.3 Description of questionnaire

There were a total of 17 questionnaires answered. The intention of the questionnaire was to use all software developers, irrespective of their work level. The individuals had to have, had past or current software development experience for an unspecified length of time. The actual questions asked were based on the 12 agile principles. This was to measure the state of pair programming practice and team outcomes experienced whilst using agile. Below is an explanation of the sections that are included in the questionnaire:

Beginning of the questionnaire:

This section consists of questions that are basic and refer to the participants themselves: demographic information about the participants, their academic background with regards to programming and their work experience. The aim of this section is to collect biographical information of participants to provide more information about the individuals in the sample.

Section A consists of two question (A.1-A.2). This section is linked to the theme of the use of pair programming. The intention of these statements is to understand the duration of prior and current pair programming experience. The subsequent sections are related to the three constructs of the Theory of Planned Behaviour. Section B consists of 20 questions (B1.1 – B4.9). This section is linked to the theme of 'attitudes' that is one of the factors in the Theory of Planned Behaviour. Section C consists of six questions (C1.1 – C2.3). This section is linked to the theme of 'subjective norm' that is one of the factors in the Theory of Planned Behaviour. Section D consists of seven questions (D1.1 – D2.5). This section is linked to the theme of 'perceived behavioural control' that is one in the factors of the Theory of Planned Behaviour. Section E consists of three questions (E1.1 – E1.3). This section is linked to the theme of 'behavioural intent' that is one of the factors in the Theory of Planned Behaviour.

End of questionnaire:

This is the last section of the questionnaire and allows participants to freely express their opinions about the questions that may have hindered them in completing the questionnaire. It allows them to comment on other points which were not raised in the questionnaire.

(Refer to Appendix B: Questionnaire Form for the full questionnaire)

4.6.4 Measurement: survey instrument

A five-point Likert Scale was used to measure the responses from the questionnaires. The one- to five-point rating scale was used to categorise responses (strongly disagree (1) to neutral (3) and strongly agree (5)). The variables used as answers to the questions are intended to establish the strength/weakness of the factors that influence the use of pair programming in an agile industry (Uma Sekaran, 2013).

4.6.5 Description of interviews

The interviews were categorised into interviews for senior staff and interviews for developers. The targeted participants were individuals with past or current software development experience. Senior staffs were classified as those individuals currently in a senior role (senior developer, team leader, manager). Junior and intermediate developer were classified as developers. The contact person from

the company was the Human Resource representative, who had provided a list of individuals to be used for the interviews. There were ten completed interviews as the company limited the study to only ten interview participants: five from senior staff and five developers. Below is an explanation of the sections that are included in the interview sheet:

There were two interview schedules, one for the senior staff and one for the developers. The questions asked were categorised according to the constructs of the Theory of Planned Behaviour viz; attitude, subjective norms, perceived behavioural control and behavioural intention.

(Refer to Appendix C: Interview Schedule for Senior Staff and Appendix D: Interview Schedule for Developers for the full interview schedule)

4.6.6 Measurement: interviews instrument

The questions asked in the interviews were open-ended so that the developers can expand on their answers. This gave the researcher a chance to ask further questions if the answers were not detailed enough the first time (Sekaran & Bougie, 2013).

4.6.7 Pilot testing of research instruments

The purpose of pilot testing the research instruments was to obtain feedback from a sample of individuals to improve the standard of the interviews and questionnaire. The pilot study involved evaluating aspects of the questionnaire and interviews such as reliability and validity of the instruments. The pilot process allows individuals to identify any errors or misleading questions and helps the researcher practice the exercise (Zaza et al., 2000).

The individuals used in the pilot study belonged to a different software development company and were currently involved in software development. The developers had minimal exposure to pair programming. The questionnaire was tested on two software developers, as were the interviews: one was a developer and the other a senior developer. The questionnaire sheet provided was the same as the one used in the actual study (Appendix B). The last section within the questionnaire allowed the respondents to express their opinions. The interview schedule used was also the same as the one used in the actual study (Appendix E.1, E.2 and E.3). The questionnaires were distributed personally to the respondents at their company's premises. Each respondent was requested to write comments or note any difficulty experienced when engaging with the research instruments. The respondents were also given the opportunity to freely express their views towards the end of the questionnaire. The individuals did not note any issues with the instruments.

4.6.8 Administration of the research instruments

The company used in this research consists of a small population of software developers that has been exposed to the use of pair programming. The developers understood pair programming well and were currently using it as a software development approach. The first step in administering the instruments was to provide a brief overview of the study. This was emailed to the participants so that they could have an understanding of the purpose and intention of the study. The researcher was on the company's premises for the full day on Friday 6 October 2017 when five interviews were scheduled, with an hour scheduled to administer the list of interview questions for each individual. At the start of the day the questionnaire was circulated to 17 respondents. The researcher personally conducted the interviews during the course of the day. By the end of the day, the researcher had collected all 17 completed questionnaire sheets. In addition, all five scheduled interviews were completed. As the remaining participants were not always on the company's premises, the remaining five interviews were conducted telephonically on Friday 27 October 2017.

The instruments were personally administered. This was to ensure that a rapport was established with the participants and to immediately clarify any doubts, thus ensuring a 100% response rate (Sekaran & Bougie, 2013).

4.7 Description of Reliability and Validity

"It is suggested that reliability is a necessary, but insufficient, condition for validity in research; reliability is a necessary precondition of validity" (Cohen et al., 2007, p. 179). In research, the validity is to ensure that the instruments are measuring what they are intended to measure. Reliability will be established by checking if the research instrument is consistent: if the same instruments were to be used then similar data should be achieved (Sekaran & Bougie, 2013).

4.7.1 Validity

This study makes use of content and criterion validation. This indicates that the instrument measures the adequate and representative set of items (Sekaran & Bougie, 2013). The focus is on the concepts within the domain, to ensure that the concepts are delineated. "Content validity is to ensure that the measure includes an adequate and representative set of items that tap the concept" (Sekaran & Bougie, 2013, p. 226). The variables used in this study are attitude, subjective norm, perceived behavioural control and intention. The validity of qualitative methods was achieved by having close collaboration with participants and company's human resource representative. This helped to gain context to the company and their process on practices (Creswell & Miller, 2000).

Validation refers to differentiating individuals based on specified criteria (Sekaran & Bougie, 2013). The questionnaire makes use of a Likert-scale for responses and allows individuals to score their responses differently. Interviews make use of open-ended questions to assess the behaviour of

software developers. The validity of the qualitative and quantitative instruments ensures the scope or depth of the data collected (Cohen et al., 2007).

4.7.2 Reliability

Reliability ensures that there is consistency, or equivalency, with data instruments. The Cronbach Alpha test is used to ensure that the questionnaires and measurements adopted in the study are applied in a consistent manner, so that the data that is generated is accurate. This is used as a means of quality control to indicate that there is reliability of the data, thereby eliminating uncertainty (Cohen et al., 2007).

4.7.2.1 Reliability results

The Cronbach's Alpha test was used to test the reliability of this study. This test will measure how closely related the conceptual framework constructs are, to ensure consistency. An alpha value >.7 indicates a reliable measure. Table 3 (below) shows the statistical results from the Cronbach Alpha test. The first column in the table indicates the conceptual framework constructs and the second column displays the measurement variables per construct (Pallant, 2001). The third column indicates the question numbers that fall into the categories. (Appendix B can be referred to for the actual questions.) The last column displays the results of the Cronbach Alpha test, which indicate whether the variables are reliable or not; as the results must be >.7 to be reliable.

Table 3: Alpha Results of Reliability using Cronbach Alpha test

Constructs	Sub-Themes per Construct	Survey Question No	Alpha Value
Attitude		B1.1 - 5.4	0.912
	Change	B1.1 - 1.3	0.701
	Learning	B2.1 - 2.3	0.894
	Collaboration & Participation	B3.1, 3.3- 3.5	0.568
	Pair Programming	B4.1 - 4.9	0.913
	Developers preference of developing with pair programming versus individual programming	B5.1 - 5.4	0.796
Subjective norms		C1.1 - 2.3	0.8
	Senior staff influence	C1.1 - 1.2	0.828
	Peer Influence	C2.1 - 2.3	0.804
Perceived Behavioural Control		D1.1 - 2.5	0.758
	Resource	D1.1 - 1.2	0.882
	Skills	D2.1 - 2.2, 2.4-2.5	0.769
Behavioural Intent		E1.1 - 1.3	0.848

All the constructs scales mentioned above are reliable (i.e. >.7), except for the sub-theme of collaboration and participation within the attitude construct (Alpha = 0.568). This result could be low due to the small size of the population. More caution will be taken when reporting results for the sub-theme collaboration and participation.

4.8 Data Capturing and Editing

The completed questionnaire sheets were filed and stored in a safe location. The results of the questionnaire were captured onto a Microsoft Excel file for statistical analysis.

The responses from the interviews conducted were coded and categorised according to the constructs of the conceptual framework. There was additional categorisation over and above the constructs of the conceptual framework: these were codes generated dynamically by each response. The coding was maintained in a codebook to ensure consistency in categorising.

The interviews from both senior staff and developers were recorded, transcribed and stored on a Microsoft Word document within the NVivo version 11.0 software tool. The transcription was conducted by a qualified typists; thus ensuring that the transcription was of a high quality. The transcriptions were also reviewed by the researcher against the audio recording to ensure that the transcriptions were accurate. The interviewer did not take down notes while conducting the interview due to inexperience, and the need to focus on facilitating the interview. Therefore, the non-verbal cues arising during the interview were not captured. However, the voice tone in the audio recording has provided more information.

4.9 Data Analysis

The qualitative data were analysed by using NVivo version 11.0 and the quantitative data were analysed with assistance of the SPSS (Statistical Package for Social Sciences) version 11.0. The details of the analysis process are explained below.

4.9.1 Quantitative analysis

The following tests were used to analyse the quantitative data:

- Descriptive statistics, including means and standard deviation, were used where applicable. They were used to identify frequencies in the demographic data that are represented in tables or graphs.

- The Chi-square goodness-of-fit-test is "A univariate test, used on a categorical variable to test whether any of the response options are selected significantly more/less often that the others" (Pallant, 2001). This test was used to compare the expected proportions with the observed

proportions from the actual study (Pallant, 2001). This test was used for question A.1 in the questionnaire. This dealt with the length of use of pair programming.

- Spearman and Pearson's correlation is a correlation test that determines the strength and direction of the correlation/relationship between two variables. The Spearman correlation measures how variables or rank orders are related. Pearson's correlation coefficient is a measurement of linear association (Pallant, 2001). The correlation was first conducted on subscales, and then with the full constructs of the conceptual framework.

- The one sample t-test was used because the population size was small. This test compares the mean score on a continuous variable to identify agreements or disagreements of the scalar value (Pallant, 2001). This test was used for question A.2, and Sections B, C, D and E in the questionnaire to determine the significant agreement/disagreement of the statements asked.

4.9.2 Qualitative analysis

NVivo is a text analysis software tool. This tool is used to organise and analyse qualitative data from individual interviews. NVivo assists in generating codebooks that are made up of nodes/codes that reflect portions of texts from transcriptions that can be categorised under specific themes (nodes). The constructs of the conceptual framework (attitudes, subjective norms, perceived behavioural control and behavioural intention) were the deductive nodes. However, as each transcription of each interview was analysed, other inductive nodes were generated. The responses from the interviews will be presented as direct quotes by the participants without editing the grammar therefore maybe grammatically correct (refer to Appendix A: Node Report from NVivo for full list of nodes used)

4.10 Ethical Consideration

In the process of collecting data, many ethical considerations were taken into account. These ethical considerations were intended to maintain the integrity, validity, confidentiality and accuracy of data and protect the participants. The company did specify certain conditions for this study such as the name of the company not to be reported and only a maximum of 10 interviews were allowed to minimise disruption on work produced. The developers' confidentiality was respected as all responses were submitted anonymously, and no cultural or racial information was requested. The information that was given by participants is stored in an excel sheet that is strictly confidential. Ms Rose Quilling, the supervisor for this dissertation, has limited access to this excel sheet and also takes responsibility for preserving the confidentiality of the documents after completion of the project. All questions contained in the questionnaire are clear and do not misguide the participants or misinterpret the nature of the study; the purpose and scope of the study is clearly stated at the beginning of the questionnaire. During the collection of the data, no responses were elicited by force; and the self-esteem or self-respect of participants was, at no time, violated. The rights of the participants of both the

questionnaire and interviews were protected by the ethical clearance process. The Humanities and Social Sciences Research Ethic Committee (UKZN) has reviewed and approved this study, which was assigned ethical clearance number HSS/1664/017M (refer to Appendix I for copy of ethical clearance approval letter). In addition, each participant was requested to complete and sign an informed consent form if they agreed to participate in the study. The content of the form covers the terms of anonymity, who will keep the data, the length of time the data will be kept, the purpose of the study and contact details to voice any concerns (refer to Appendix B).

4.11 Conclusion

This study makes use of the mixed method approach that adopts questionnaires and interviews as research instruments. The questionnaire and half the interviews were personally administered, while the remainder of the interviews were conducted telephonically. The researcher makes use of the NVivo tool to analyse the qualitative data and the SPSS tool to analyse the quantitative data. In order to ensure the validity of the data questions in the questionnaires, the Likert scale and open-ended questions in the interviews were used. The Cronbach Alpha test was adopted to ensure reliability. In addition to ensure validity of qualitative data the trustworthiness of validity is adopted. The participants' rights have been protected by signing the informed consent form that ensures the protection of anonymity. The study has also been reviewed and given ethical clearance. The next chapter will discuss the basic analysis and interpretations of the quantitative data in order to draw related conclusions.

Chapter 5: Quantitative Data Analysis

5.1 Introduction

This chapter will present the results of the statistical analysis of the quantitative data. The data obtained in the questionnaires is used to provide a broad understanding which will be explored in further detail in the interviews. Therefore, the quantitative data is presented first. The results will be interpreted according to the constructs of the conceptual framework: attitude, subjective norm, perceived behavioural control and behavioural intention. This provides a better flow and understanding of the data which link the research questions and objectives, to the conceptual framework. The results are presented and categorised according to the constructs of the conceptual framework. In addition, the results of reliability tests and correlation tests are presented. The following statistical tests were used in the data analysis:

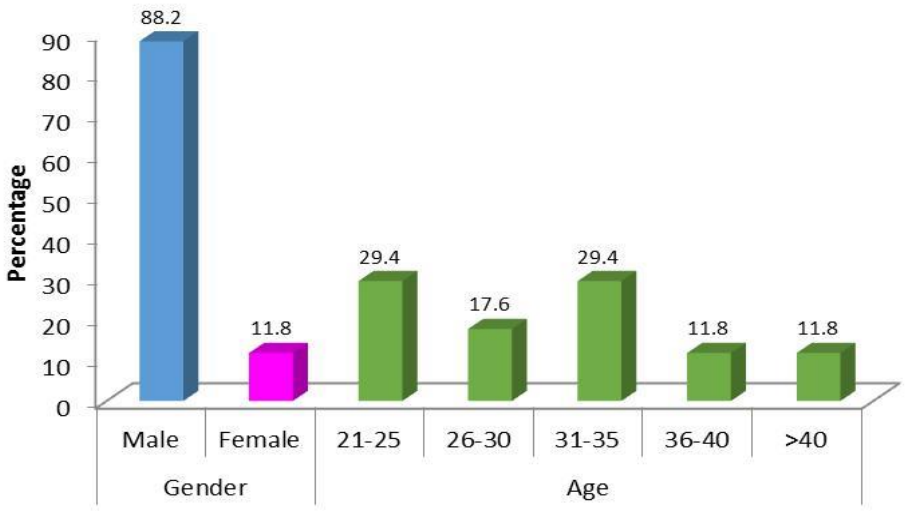
- Descriptive statistics including means and standard deviations were used where applicable. Frequencies are represented in tables or graphs. These results provide an indication of the sample dynamics in this study.
- The chi-square goodness-of-fit-test is used to establish if responses were significantly chosen more or less often, and is used on categorised results (Pallant, 2001). This test is used to ascertain if the actual results obtained in the study differ from those described in the literature.
- The one sample t-test is used to indicate if a mean is significantly different from the fixed/scalar value. This test will use the mean value 3, on the Likert scale, to represent the neutral category (Pallant, 2001). The t-test results are then compared with the mean value. Results less than the mean value indicate disagreement, while values higher than the mean indicate agreement, with the statement.
- The Cronbach Alpha test is used as a reliability test. This is used to establish if the factors are reliable and can be reported (Pallant, 2001).
- Pearson's correlation test is used to check the correlation between variables or rank orders (Pallant, 2001).
- Spearman's correlation test is also used to check correlation between the strength or direction of associated variables. This test is used when data is scaled (Pallant, 2001).

5.2 Description of the Sample Demographic Details

Below is a description of the sample which answered the questionnaires. The information below represents the demographics and dynamics of the sample population used in this study, and should explain any limitations of the study due to the demographics involved.

Gender and Age

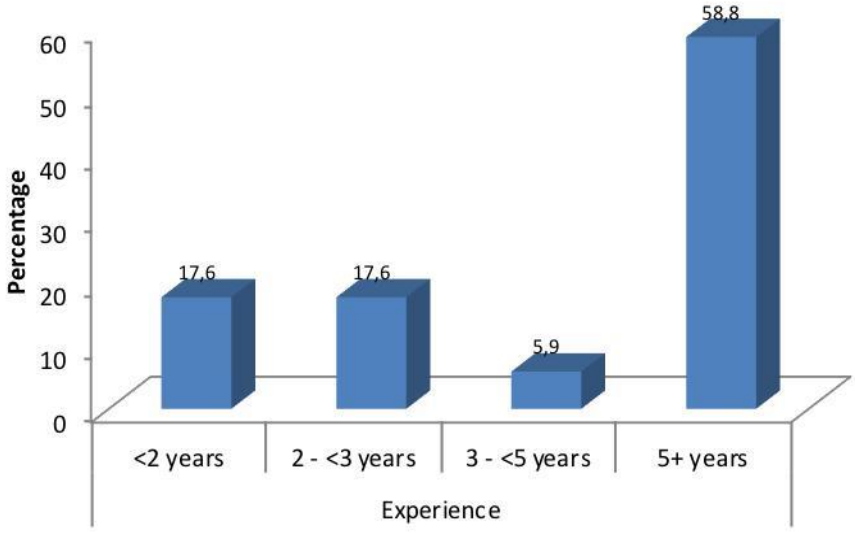
Figure 3: Gender and Age



As is shown above, the dominant age groups are in the ranges 21-25 and 31-35. There were 15 males (88.2%) and 2 (11.8%) females in a total of 17 respondents. This company employs more males than females and therefore the perception of pair programming from a female perspective cannot be adequately researched.

Experience

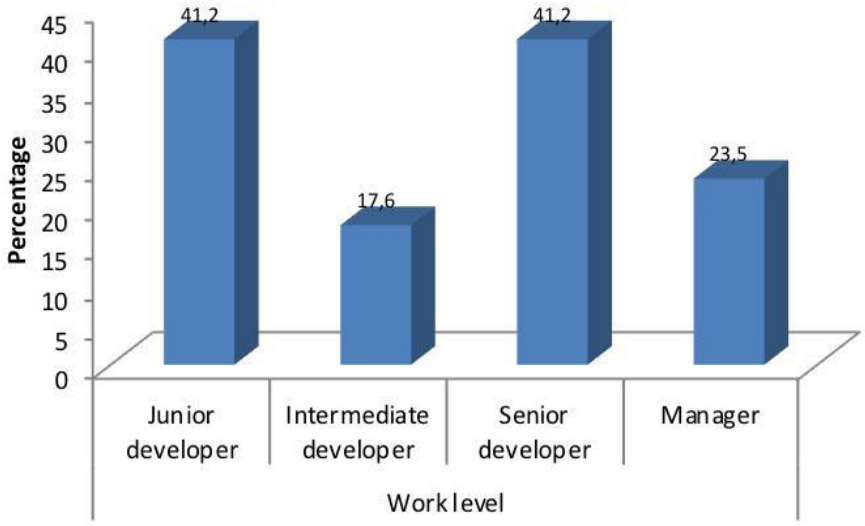
Figure 4: Years of Experience



From the above, it can be seen that 58.8% (10 people) of the respondents had more than five years' experience, so the software developers are mostly experienced. There was an equal representation of software developers with <2 and 2-<3 years 17.6% (3 people) of software development experience.

Work Level

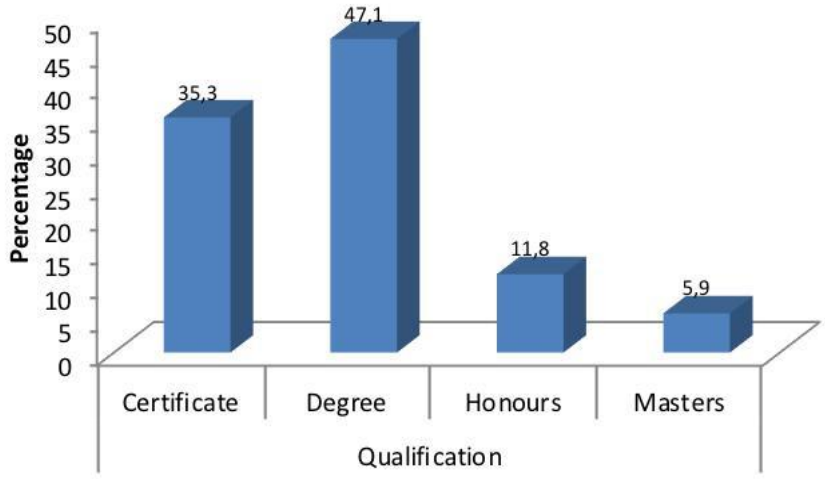
Figure 5: Work Level



The graph, above, shows an equal distribution of junior and senior software developers (41.2%, 7 people each) which indicates that there is equal insight from both experienced and less experienced software developers. This indicates no bias in the data towards a specific level of experience. There are 23.5% (4 people) managers and the rest are intermediate developers, of which there are only three. It should be noted that the employees in the company researched had ticked more than one role that is applicable to them as they perform different roles within different teams. Therefore an individual can perform a number of different roles in various teams. This means the total number of participants identified on figure 5 will not equal the total number of participants of the study.

Qualifications

Figure 6: Qualifications



From the above, it can be seen that 64.8 % (11 people) of the software developers have a degree (Bachelors, Honours and Masters). Of the software developers in this sample, 35.3% (6 people) have a certificate and 17.7% (3 people) obtained an advanced Honours or Masters qualification.

The demographic information shows that the study sample consists mainly of males (17 males; 2 females); so this study can only draw limited conclusions about the perceptions of females in particular, and males. Previous studies have noted the lack of woman in this industry, which is a global phenomenon. The percentage of women in the computing industry has been drastically decreasing since the early years, that is since 1991 (Ashcraft, McLain, & Eger, 2016). The dominant age groups within this population were between 21 and 25, and from 31 to 35 years old. Ashcraft et al. (2016) indicate that females in the age group 24 to 34 feel 'stalled' in computing occupations, which could explain the low numbers in this sample. The least represented age groups were 36 to 40 and over 40 years old. Most of the software developers had more than five years' experience in the industry. There was an equal representation of junior and senior developers 41.2% (7 people) work levels and intermediate developers were the least represented work level. Most of the developers have obtained some degree qualification; and there was one individual with a Master's degree.

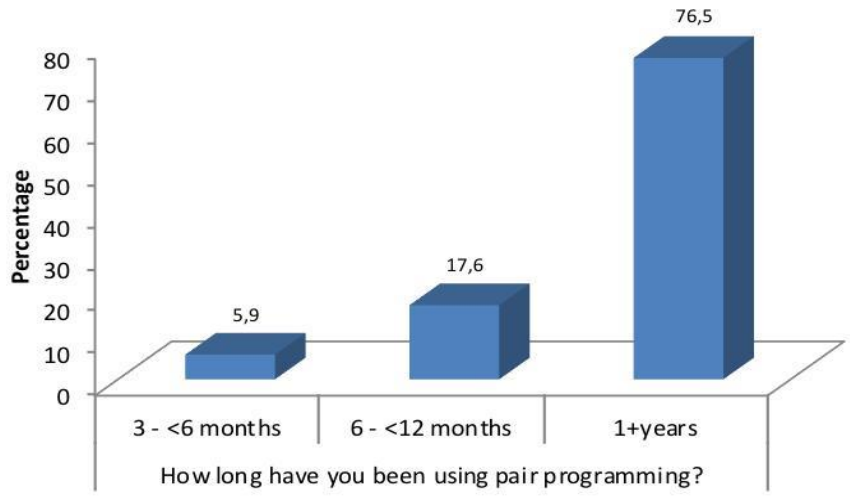
5.3 Use of Pair Programming

The use of pair programming is the underlying behavioural focus of this study, and is also the main variable of the conceptual framework. The objective of the survey is to determine software developers' use of pair programming within agile software development. The analysis of the responses to the questions in this section makes use of the chi-square goodness of fit test. This test is used to determine the likelihood that the observed distribution is due to chance. The goodness of fit

test is used as it will measure the observed, versus expected, distribution for independent variables. Therefore, the results will indicate if any response occurs (statistically) significantly more often more than the other response (Pallant, 2001).

The first question of this section (A1.2) asks: "How long have you been using pair programming?" The responses are shown in Figure 7, below.

Figure 7: Responses to question A1.2



In the company that was researched, 76.5% (13) of the respondents have been using pair programming for at least one year. The statistical analysis determined a significance of $\chi 2$ (2) = 14.588, p=.001 (p<.005). In addition the statistical t-test indicates significant agreement (M=3.88, SD = 1.054) i.e. that pair programming is currently being used in projects, t (16) = 3.453, p=.003 (p<.005). These results suggest that the continued use of pair programming and sustained current usage indicates positivity about using pair programming. (Appendix F: Survey Statistical Results, Section A).

The second question (A2.1) states: "I currently use pair programming in my projects". The analysis of the responses to this statement makes use of the one sample t test. This test calculates whether a mean score is significantly different from a scalar value (Pallant, 2001) of 3: a result of t < 3 indicates that the person disagrees with the statement; t > 3 indicates agreement. This analysis shows that there is significant agreement (M=3.88, SD = 1.054) that pair programming is currently being used in projects, t (16) = 3.453, p=.003 (p<.005) (Appendix F: Survey Statistical Results, Section A).

The data obtained from the sample population surveyed in this study indicates that the majority of the software developers in this context are significantly positive about pair programming and most of these software developers have used pair programming for more than one year (76.5%). A small

percentage (5.9%) of software developers has only used pair programming for between three and six months. This indicates that more than half of the software developers have used pair programming for an extended period of time and understand the terms, dynamics and processes of pair programming. Hence, they are able to relate to, and understand, the remainder of the questions within the survey. This also indicates that their input is representative of experienced pair programmers as most of the software developers have used pair programming for more than a year and are currently using pair programming.

Previous studies indicate that the adoption of pair programming is relatively low (Doyle et al., 2014). However, the results of this study indicate that pair programming is used more in this company than was shown in previous studies. Moreover, it has been used for an extended period of time by individual developers in this case-study company. These results were expected because the case study was purposively selected because pair programming was being used within an agile approach, within the company (refer to section in Chapter 4: Discussion of the research site). The results of this case study are thus different to what was noted in previous studies, which mention that pair programming is the least adopted agile practice (Doyle et al., 2014). This study focuses on the perceptions of those who are using pair programming, even though they may be in the minority in the industry.

5.4 Attitude

This factor 'attitude' is one of the main construct of the conceptual framework. This factor relates to the research question*: How do attitudes correlate to software developers' use of pair programming within an agile software development methodology?* The objectives of this construct are to assess the role of *attitudes* towards using pair programming within agile software development by determining:

- the attitudes of software developers towards development when using pair programming within agile for software development
- the attitudes of software developers towards collaborating in a team using pair programming within agile for software development

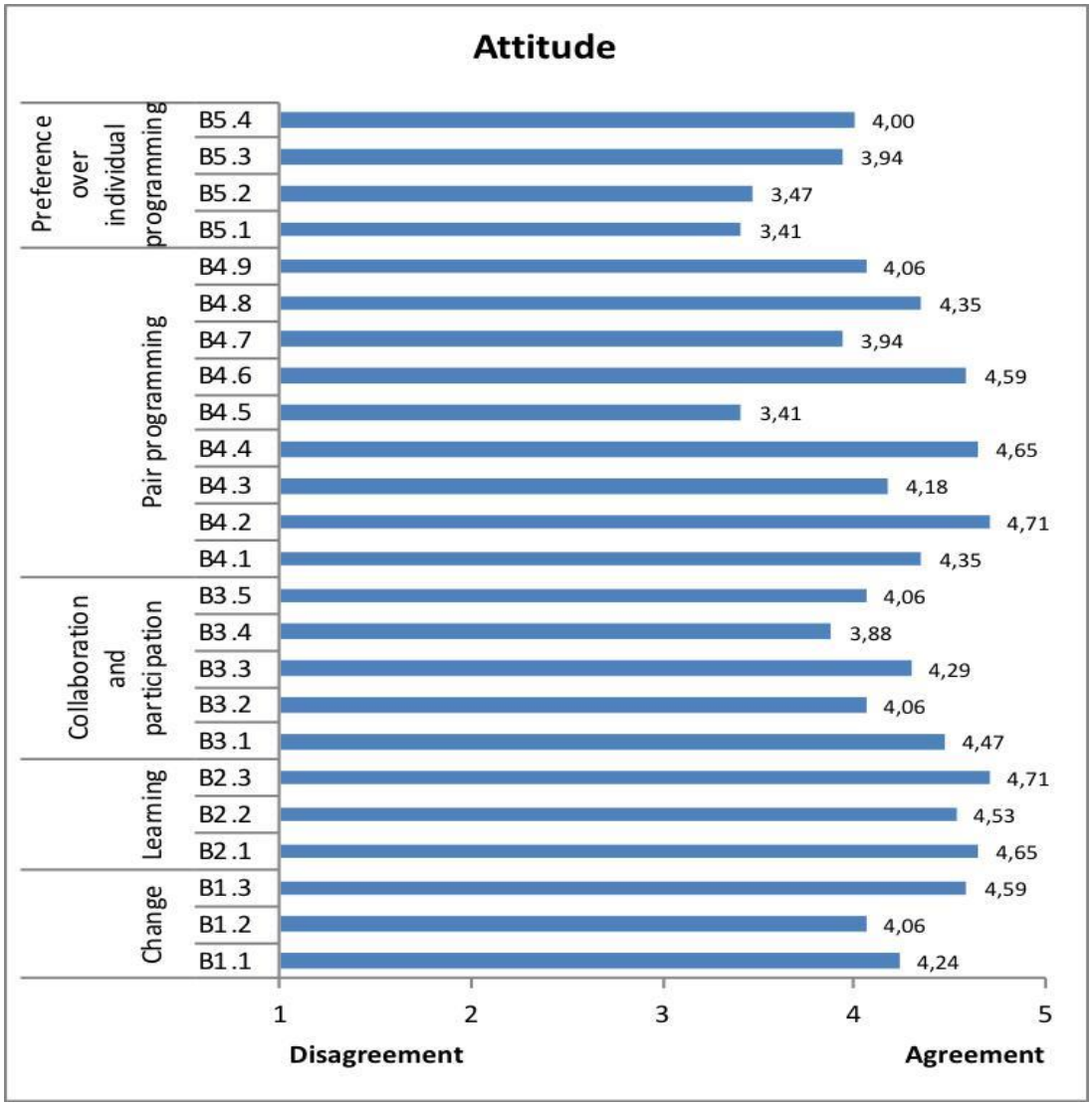
(Refer to Appendix F: Survey Statistical Results, Section B for the full statistical results)

Figure 8: Survey: detailed questions within Section B (Attitude)

B1.1 I welcome change and can cope when changes are applied to methodology at any stage of development, even late in development
B1.2 I welcome change to the process, even when it's the first time I experience something
B1.3 I am happy to change the way I do programming if it results in better output
B2.1 I am interested in learning new things about design issues in software development
B2.2 I will gladly learn a new method of programming
B2.3 I will gladly learn about new programming tools or techniques
B3.1 I am happy to follow the process defined and agreed upon by the team
B3.2 I trust my team members to make decisions on my behalf
B3.3 I am motivated by my team members to participate in the development processes
B3.4 I prefer working in a team to working alone
B3.5 I work well with others on a task
B4.1 I feel productive when using pair programming
B4.2 I feel pair programming empowers knowledge sharing
B4.3 I feel pair programming improves my self-motivation
B4.4 I believe developers find and implement better code solutions by using pair programming
B4.5 I feel pair programming reduces documentation
B4.6 I feel errors are found earlier when using pair programming
B4.7 I feel pair programming reduces the need for review/inspection of code
B4.8 I feel pair programming produces high quality code
B4.9 I feel pair programming produces high quality code faster
B5.1 I prefer using the pair programming technique over individual programming techniques
B5.2 I feel more productive using pair programming than programming on my own
B5.3 I feel pair programming is effective in reducing the software development effort compared to individual programming
B5.4 I feel the software delivered when using pair programming technique is of a higher quality than individual programming

The main variable, 'attitudes', is further broken down into measurement variables: attitude towards change, attitude towards learning, attitude towards collaboration and participation, attitude towards pair programming, and developer's preference for developing with pair programming versus individual programming. The analysis of the responses to the questions in this section makes use of the one sample t-test, which calculates whether the mean scores are significantly different from the scalar value of 3 (neutral).

Figure 9: Survey: mean responses to Section B (Attitude)



Attitude towards change

The questions B1.1 - B1.3 within the questionnaire relate to attitude towards change. The results of the analysis of the responses to these questions are as follows:

- B1.1- There is significant agreement (M=4.24, SD = .752) that software developers welcome change and can cope when changes are applied to the software development methodology at any stage of development, even late in development, t (16) = 6.769, p = .000 (p <.005)
- B1.2- There is significant agreement (M=4.06, SD = .659) that software developers welcome change to the process, even when it's the first time of experiencing something, t (16) = 6.628, p = .000 (p <.005)

- B1.3- There is significant agreement (M=4.59, SD = .507) that software developers are happy to change the way they do programming if it results in better output, t (16) = 12.908, p = .000 (p <.005)

The results of the statistical analysis of attitudes towards change indicated that change and adjustment are part of the accepted culture of software developers in this company. They are able to cope with change and can adjust easily to change. Hence, their attitude towards change is positive. In question B1.3: 'I am happy to change the way I do programming if it results in better output', all the responses fall into the category of 'agreement' (seven respondents agree; ten respondents strongly agree). This suggests that software developers value their output more than the potential discomforts of having to change, so if changing will improve their output, they welcome the change as they value their output more.

Previous studies indicate that an agile environment is prone to changes, however not every individual can cope or embrace change (Asaria et al., 2014). Therefore by having a positive attitude towards change, there is the willpower to overcome the difficulties of change and cope in an ever-evolving technological environment (Balbes, 2014).

Attitude towards learning

The questions B2.1 – B2.3 within the questionnaire relate to attitudes towards learning (see Figure 9 above). The results of the analysis of the responses to these questions are as follows:

- B2.1- There is significant agreement (M=4.56, SD = .493) that the software developers are interested in learning new things about design issues in software development, t(16) = 13.786, p = .000 (p <.005)
- B2.2- There is significant agreement (M=4.53, SD = .624) that the software developers will gladly learn a new method of programming, t (16) = 10.101, p = .000 (p <.005)
- B2.3- There is significant agreement (M=4.71, SD = .470) that the software developers will gladly learn about new programming tools or techniques, t (16) = 14.976, p = .000 (p <.005)

The overall results show that the respondents have a positive attitude towards learning. Since agile involves frequent response to change, the ability to improve knowledge is vital. Since attitudes towards learning are positive, this could suggest that if uncertainty is experienced when using pair programming, the developers will have a positive attitude towards finding out and understanding ways to implement pair programming.

Previous studies mention that pair programming is commonly used as a method to upskill an inexperienced developer (Tsyganok, 2016). Therefore, the more positive the attitude towards learning,

the more likely that with exposure, experience and understanding is built, which contributes to improving the quality of work.

Due to the high number of very positive responses, there could be bias, in the form of social desirability, in the data. This means that the respondents' answers could be influenced by their wanting to provide the socially desirable responses in order to conform with behaviour that they believe is generally accepted around them (Podsakoff, MacKenzie, Lee, & Podsakoff, 2003). In this case it would mean they believe they should be eager to learn new things in their working environment as software developers.

Attitudes towards collaboration and participation

The questions B3.1 – B3.5 within the questionnaire relate to attitudes towards collaboration and participation (see Figure 9 above). The results of the analysis of the responses to these questions are as follows:

- B3.1- There is significant agreement (M=4.47, SD = .624) that the software developers are happy to follow the process defined and agreed upon by the team, t(16) = 9.713, p=.000 (p <.005)
- B3.2- There is significant agreement (M=4.06, SD = .659) that the software developers trust the team members to make decisions on their behalf, t (16) = 6.628, p = .000 (p <.005)
- B3.3- There is significant agreement (M=4.29, SD = .686) that the software developers feel motivated by team members to participate in the development processes, t (16) = 7.778, p=.000 (p <.005)
- B3.4- There is significant agreement (M=3.88, SD = .686) that the software developers find working in a team more favourable than working alone, t (16) = 3.665, p=.002 (p<.005)
- B3.5- There is significant agreement (M=4.06, SD = .659) that the software developers feel that they work well with others on a task, t (16) = 6.628, p = .000 (p <.005)

The overall results show that the attitude towards collaboration and participation is positive. The software developers appear to have a positive attitude to working in a team and are able to work well with others. They also consider the team's perspective when making a decision; they can work well in a team environment and find a team environment motivating. Positive attitudes towards collaboration and participation build stronger team relationships and improve self-morale (Begel & Nagappan, 2008). The positive attitude will impact the behaviour of developers by making them more willing to collaborate and participate. The software developers also indicate that they find working in a team more favourable than working alone, and this could be seen to contradict previous studies which indicate that software developers program more efficiently when working individually than when

using pair programming (Kim et al., 2010). This is linked to the inconclusive response to question B5.1 and will be discussed later. The more developers collaborate and participate, the more this will improve the sharing of knowledge and understanding, leading to improved productivity. The responses to question B3.5 indicate that the developers feel positive about working in a team, and this could motivate them to use pair programming more than individual programming. Asaria et al. (2014) mentioned that the more individuals accept another person in their space, then the more interactive they would be in team work, leading to positive benefits.

Attitudes towards pair programming

Questions B4.1 – B4.9 within the questionnaire relate to attitudes towards pair programming (see Figure 9 above). The results of the analysis of the responses to these questions are as follows:

- B4.1- There is significant agreement (M=4.35, SD = .786) that the software developers feel productive when using pair programming, t (16) = 7.098, p=.000 (p<.005)
- B4.2- There is significant agreement (M=4.71, SD = .470) that the software developers feel pair programming empowers knowledge sharing, t (16) = 14.976, p=.000 (p<.005)
- B4.3- There is significant agreement (M=4.18, SD = .883) that the software developers feel pair programming improves self-motivation, t (16) = 5.494, p=.000 (p<.005)
- B4.4- There is significant agreement (M=4.65, SD = .493) that the software developers believe developers find and implement better code solutions by using pair programming, t(16)= 13.786, p=.000 (p<.005)
- B4.5- There is not a significant result although the values are indicative of disagreement (M=3.41, SD = 1.004) with the statement that software developers feel pair programming reduces documentation, t (16) = 1.692, p=.110 (p>.05). The result should be considered inconclusive.
- B4.6- There is significant agreement (M=4.59, SD = .795) that the software developers feel errors are found earlier when using pair programming, t (16) = 8.235, p=.000 (p<.005)
- B4.7- There is a significant, slight disagreement (although the standard deviation is large which depicts a high level of variability) (M=3.94, SD =1.298) that the software developers feel pair programming reduces the need for review/inspection of code, t (16) = 2.991, p=.009 (p>.05)
- B4.8- There is significant agreement (M=4.35, SD = .702) that the software developers feel pair programming produces high quality code, t (16) = 7.948, p=.000 (p<.005)
- B4.9- There is significant agreement (M=4.06, SD = 1.088) that the software developers feel pair programming produces high quality code faster, t (16) = 4.012, p=.001 (p<.005)

The use of pair programming tends to provide admirable benefits to the software developer, which results in a positive attitude. The above results concur with previous studies that mention the benefits realised when using pair programming: for example, improving productivity, quality of work, self-esteem and knowledge empowerment (Begel & Nagappan, 2008; Cockburn & Williams, 2000; Williams, 2010). The developers were in agreement that pair programming makes them feel productive, improves their self-motivation and empowers knowledge sharing. This assists in motivating the software developers to use pair programming. Some other benefits are improved quality of work and finding better solutions. As Williams (2010, p. 5) states, with pair programming "teams are motivated to institute pair programming due to the positive effects of knowledge sharing like instances where someone is on leave, if one person is struggling there is another to assist". This also helps to avoid dependence on one individual since everyone on the task will have knowledge. Therefore, the increased knowledge will help speed up the task and make an individual feel more motivated, hence improving productivity.

It was noted that the responses to question B4.5 are inconclusive. This contradicts previous studies which indicate that in an agile environment the focus is more on communication as an approach to finalise rules and decisions, instead of the traditional way of using documentation (Doyle et al., 2014). As pair programming involves constant communication between the driver and navigator, this should reduce the amount of documentation produced (Williams et al., 2000). However, the analysis of the survey responses shows that the programming approach used has little effect on the documentation.

Developer's preference for developing with pair programming versus individual programming

The questions B5.1 – B5.4 within the questionnaire relate to developers' preference for pair programming versus individual programming (see Figure 9 above). The results of the analysis of the responses to these questions are as follows:

- B5.1- There is no significant result (M=3.41, SD = .870) showing whether the software developers prefer to use pair programming over individual programming, $t(16)=1.951$, $p=.069$ ($p>.05$)
- B5.2- There is significant disagreement with the statement (M=3.47, SD = .874). The software developers do not feel more productive using pair programming than programming individually, $t(16) = 2.219$, $p=.041$ ($p<.05$)
- B5.3- There is significant agreement (M=3.94, SD = .748) that the software developers feel pair programming is effective in reducing the effort required in software development, compared to individual programming, $t(16) = 5.191$, $p=.000$ ($p < .005$)

- B5.4- There is significant agreement (M=4.00, SD = .866) that the software developers feel the software delivered when using pair programming technique is of a higher quality than when programming individually, t (16) = 4.761, p=.000 (p <.005)

The overall results indicate a positive attitude towards using pair programming versus programming individually. The software developers feel pair programming reduces the amount of effort required in software development and produces a higher quality of work, than individual programming. The software developers prefer to use pair programming over individual programming because they are aware of the positive benefits of pair programming and feel it delivers a better product. In this study there was no significant indication that software developers showed any preference for pair over individual programming. In response to question B5.2, the software developers indicate that they significantly disagree that pair programming makes them feel more productive than programming on their own. However, previous studies suggest that pair programming is beneficial as a result of improved code quality and knowledge sharing, such that "even though you were not productive, you still want to pair, because the result of quality code is much better" (Williams et al., 2000, p. 6).

It was noted that the responses to question B5.1 are inconclusive, but link back to question B3.4 where responses were significantly in agreement that: 'I prefer working in a team to working alone'. This contradicts previous studies which indicate that software developers prefer to work individually rather than in pair programming (Balijepally et al., 2009; Kim et al., 2010). The "common" belief recorded in the literature could be because the majority of people in software development believe that programmers work better alone, and this may thus have become the 'dominant discourse' (Kress, 2009). Potentially, software developers may then believe that working alone is better because this is what other developers felt and said; without this being tested to validate it as truth.

5.5 Subjective norm

This factor, 'subjective norm', is one of the main constructs of the conceptual framework, Theory of Planned Behaviour. This factor relates to the research question*: How do subjective norms correlate to software developer' use of pair programming within an agile software development methodology?* The objectives of this construct are to assess the role of software development *subjective norms* towards using pair programming within agile software development by determining:

- the manager's or team leader's influence on software developers when using pair programming within agile software development
- peer influence on software developers within the organisation when software developers use pair programming within agile for software development

The main variable 'subjective norm' is further broken down to measurement variables to determine the motivating or demotivating factors when using pair programming. The measurement variables are senior staff and peer influence. The data is statistically analysed using the one sample t-test which compares results to the mean response on the Likert scale M=3.

(Refer to Appendix F: Survey Statistical Results, Section C for the full statistical results)

Figure 10: Survey: mean response to Section C (Subjective Norms)

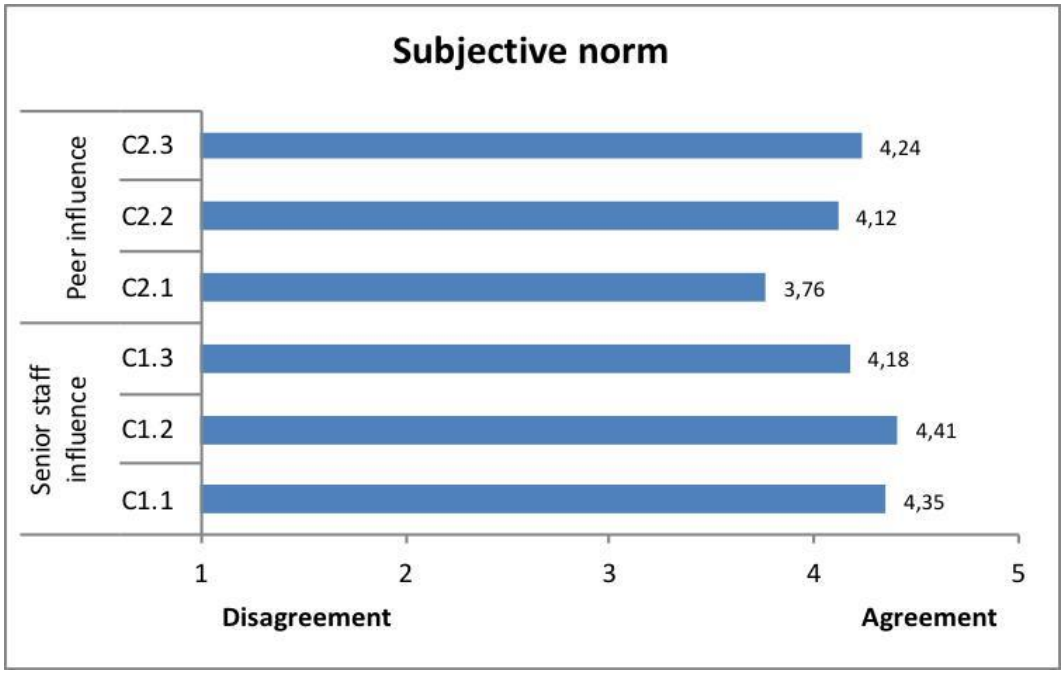


Figure 11: Survey: detailed questions within Section C (Subjective Norms)

C1.1 My superiors encourage the use of pair programming
C1.2 My superior provides formal opportunities for the developers to learn pair programming
C1.3 My superior feels pair programming improves the delivery of a product
C2.1 My peers think pair programming should be used because it reduces the amount of time spent on a task
C2.2 My peers think pair programming should be used because it improves the quality of software
C2.3 My peers like to use pair programming whenever it is applicable

Senior Staff influence

Questions C1.1 – C1.3 within the questionnaire relate to senior staff influence (see Figure 11 above). The results of the analysis of the responses to these questions are as follows:

- C1.1- There is significant agreement (M=4.35, SD = .606) among the software developers that superiors encourage the use of pair programming, t (16) = 9.200, p=.000 (p<.005)
- C1.2- There is significant agreement (M=4.41, SD = .507) among the software developers that superiors provide formal opportunities for the developers to learn pair programming, t(16) = 11.474, p=.000 (p<.005)
- C1.3- There is significant agreement (M=4.18, SD = .636) that the software developers' superiors feel pair programming improves the delivery of a product, t (16) = 7.628, p=.000 (p<.005)

The overall results show that senior staff motivates the use of pair programming. This implies that senior staff promotes the use of pair programming and facilitates its use by providing formal opportunities to adhere to, or improve on, the understanding of pair programming dynamics. The software developers feel that their superiors believe pair programming improves the delivery of a product which is another motivating factor for using pair programming. A respondent noted: "Pair programming is often most effective when pairing with a senior…" It was also noted that the ratios of junior and senior developers within this organisation are equal. Previous studies indicated that pair programming is used well as a 'training tool'. Therefore pair programming should be a good fit for the company with the equal numbers of junior and senior staff since, besides the other benefits it can provide, pair programming can serve as an in-house training process (Tsyganok, 2016).

Peer influence

The questions C2.1 – C2.3 within the questionnaire relate to peer influence (see figure 11 above). The results of the analysis of the responses to these questions are as follows:

- C2.1- There is significant agreement (M=3.76, SD = .752) that the software developers feel that their peers think pair programming should be used because it reduces the amount of time spent on a task, t (16) = 4.190, p=.001 (p<.005)
- C2.2- There is significant agreement (M=4.12, SD = .857) that the software developers feel that their peers like to use pair programming whenever it is applicable, t (16) = 5.373,p=.000 (p<.005)
- C2.3- There is significant agreement (M=4.24, SD = .562) that the software developers feel that their peers think pair programming should be used because it improves the quality of software, t (16) = 9.058, p=.000 (p<.005)

The results show significant agreement that peers feel pair programming improves the quality of software and reduces the time spent on a task. This aligns with previous studies in the literature that identify these as benefits of pair programming (Begel & Nagappan, 2008; Kim et al., 2010; Williams, 2010). Peers would like to use pair programming, which suggests that they favour it as a practice. Software developers feel that their peers are positive about using pair programming, and are therefore motivated to use it (Williams, 2006). The supportive culture of peers makes the use of pair programming more comfortable as it is perceived to be generally accepted (Asaria et al., 2014).

5.6 Perceived Behavioural Control

The factor 'perceived behavioural control' is one of the main constructs of the conceptual framework, Theory of Planned Behaviour. This factor relates to the research question*: How does perceived behavioural control correlate to software developer' use of pair programming within an agile software development methodology?* The objectives of this construct are to assess the role of *perceived behavioural control* in software development using pair programming within agile software development by determining:

- the resources/environmental influences on software developers when using pair programming within agile for software development
- the software developers' perceived behavioural control's influence on their use of pair programming within agile for software development

The main variable, 'perceived behavioural control', is further broken down into measurement variables to depict the ease or difficulty factors when using pair programming. The measurement variables are resources/environmental factors and skills/abilities. The data is statistically analysed using the one sample t-test which calculates whether the mean scores are significantly different from the scalar value of 3 (neutral) on the Likert scale.

(Refer to Appendix F: Survey Statistical Results, Section D for the full statistical results)

Figure 12: Survey: mean response to Section D (Perceived Behavioural Control)

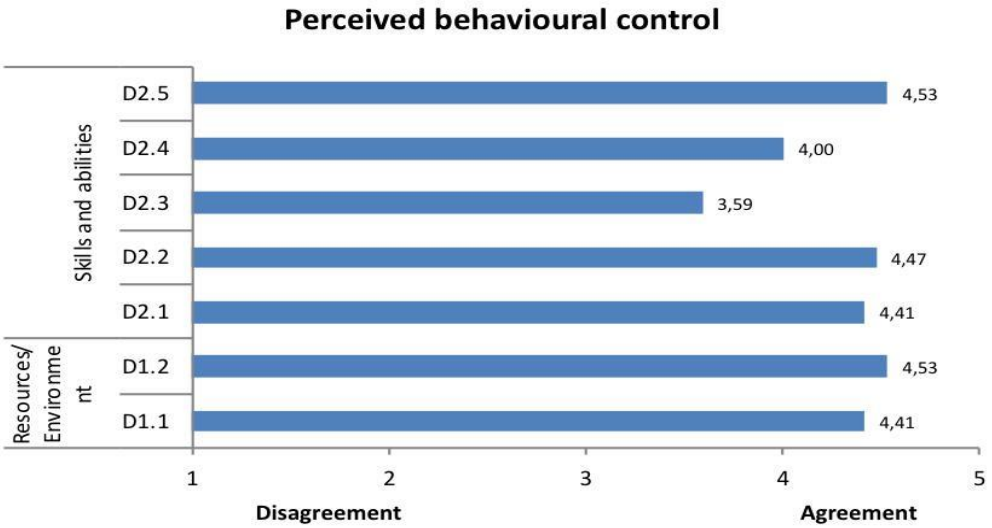


Figure 13: Survey: detailed questions within Section D (Perceived Behavioural Control)

D1.1 Pair programming can be done using the existing hardware available
D1.2 Pair programming is possible with the existing layout of the office space
D2.1 I am able to write code correctly
D2.2 I am able to test the code I write
D2.3 I have been formally taught/trained to use pair programming
D2.4 I have enough work experience and knowledge on pair programming
D2.5 I have the skills to debug faulty programming code

Resource/Environmental

The questions D1.1 – D1.2 within the questionnaire relate to perceived behavioural control (see Figure 13 above). The results of the analysis of the responses to these questions are as follows:

- D1.1 - There is significant agreement (M=4.41, SD = .507) among software developers that pair programming can be done using the existing hardware available, t (16) = 11.474, p=.000 (p<.005)
- D1.2- There is significant agreement (M=4.53, SD = .514) among software developers that pair programming is possible with the existing layout of the office space, t(16)=12.257,p=.000 (p<.005)

The results indicate that the company used in this research has sufficient hardware and an office layout which facilitates the use of pair programming. This implies that the software developers of this company do not have difficulties with hardware or layout that can hinder the use of pair programming. Therefore, software developers can easily engage with the pair programming approach and motivate the adoption of pair programming. As pair programming is vastly different from traditional programming, there is a need to have resources/environmental factors that support the dynamics of using pair programming (Williams, 2010). Since this company contains the right, and sufficient, resources, and has a supportive environment, the software developers will be able to use pair programming with little to hinder the practice (Asaria et al., 2014).

Skills and abilities

The questions D2.1 – D2.5 within the questionnaire relate to perceived behavioural control (see Figure 13 above). The results of the analysis of the responses to these questions are as follows:

- D2.1- There is significant agreement (M=4.41, SD = .507) that the software developers feel that they are able to write code correctly, t (16) = 11.474, p=.000 (p<.005)
- D2.2- There is significant agreement (M=4.47, SD = .514) that the software developers feel that they are able to test the code they write, t (16) = 11.785, p=.000 (p<.005)
- D2.3- There is no significant result (M=3.59, SD = 1.228) to indicate whether the software developers feel they have been formally taught/trained to use pair programming, t (16) = 1.975, p=.066 (p>.05)
- D2.4- There is significant agreement (M=4.00, SD = .866) that the software developers feel they have enough work experience on, and knowledge of, pair programming, t (16) = 4.761, p=.000 (p<.005)
- D2.5- There is significant agreement (M=4.53, SD = .514) that the software developers feel they have the skills to debug faulty programming code, t (16) = 12.257, p=.000 (p<.005)

This implies that the software developers within this study understand, and know well, how to program/write code correctly. Their programming expertise is good and therefore enables them to use pair programming. In addition, the software developers are skilled at writing and testing code. This suggests that the software developers are highly skilled and knowledgeable and can therefore do pair programming. There was strong agreement with the statement in question D2.5: 'I have the skills to debug faulty programming code'. This highlights that all the developers possess the software development skills necessary to review/debug or test code. This implies that the software developers generally appear to believe they have the necessary skills and abilities to perform pair programming and therefore their personal choice of programming approach would not be limited by a lack of a specific skill set.

There was no significant agreement or disagreement in the responses to question D2.3: that the software developers feel they have been formally taught/trained to use pair programming. However, in spite of the inconclusive result, and whether or not they have been adequately trained in pair programming, the software developers seem to agree that they have the skills and knowledge to be able to handle the different aspects that are required in pair programming. This could be because the majority of the software developers have more than one year experience in pair programming.

Since pair programming is very different to the traditional programming approach, it is necessary to have software developers with the right skills and sufficient skills in order to engage with the process. This builds a proactive team than can reap the benefits of pair programming by being actively involved (Asaria et al., 2014).

5.7 Behavioural Intention

The factor, 'behavioural intention', is the main focus of this study. The behavioural intention to use pair programming is the overarching factor in the sub-themes within the conceptual framework (attitudes, subjective norms and perceived behavioural control). This factor relates to the research question*: How does intention correlate to software developers' use of pair programming within an agile software development methodology?* The objectives of this construct are to assess software developers' intentions regarding the use of pair programming for current and future development within an agile industry. The data is statistically analysed using the one sample t-test, comparing results to the mean response on the Likert scale means value (3).

(Refer to Appendix F: Survey Statistical Results, Section E for the full statistical results)

Figure 14: Survey: mean response to Section E (Behavioural Intent)

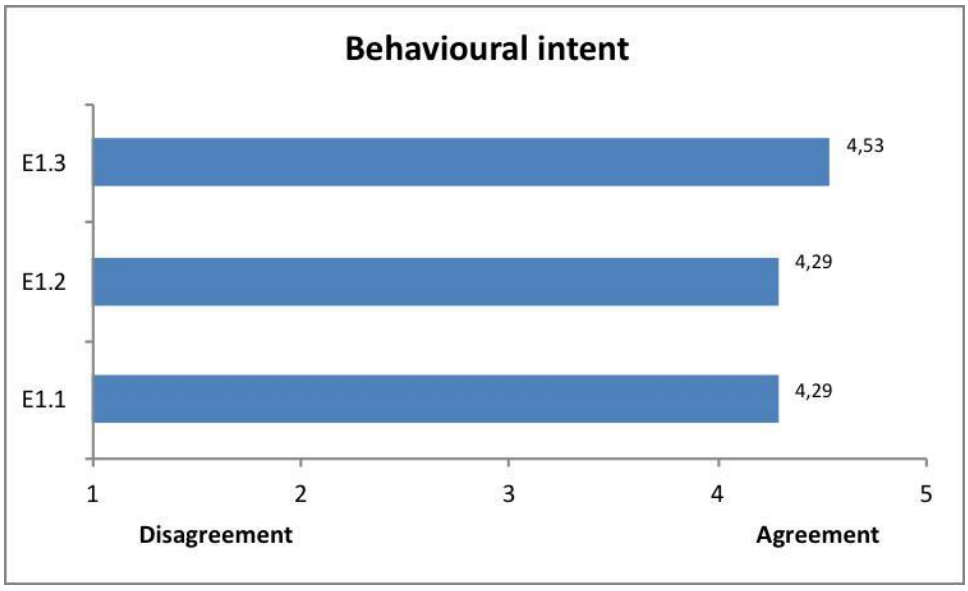


Figure 15: Survey: detailed questions within Section E (Behavioural Intent)

E1.1 Given the choice, I will use pair programming in my current project, if applicable
E1.2 Given the choice, I will increase my use of pair programming, where applicable
E1.3 Given the choice, I will using pair programming in the future, where applicable

The questions E.1 – E1.3 within the questionnaire relate to behavioural intent (see figure 15 above). The results of the analysis of the responses to these questions are as follows:

- E1.1- There is significant agreement (M=4.29, SD = .686) that the software developers feel, given the choice, they will use pair programming in their current project, if applicable, t(16)=7.778, p=.000 (p<.005)
- E1.2- There is significant agreement (M=4.29, SD = .588) that the software developers feel, given the choice, they will increase their use of pair programming, where applicable, t(16)=9.077, p=.000 (p<.005)
- E1.3- There is significant agreement (M=4.53, SD = .514) that the software developers feel, given the choice, they will use pair programming in the future, where applicable, t(16)=12.257, p=.000 (p<.005)

The analysis indicates that the software developers feel that, given the choice, they will use pair programming currently and in the future. They are also positive about increasing their current use of pair programming. This implies that the developers are satisfied with the current use of pair programming and are happy to increase its use in the future. This implies that pair programming is a technique that will continue to be used due to the benefits realised.

The software developers do use pair programming, but the results overall are inconclusive in terms of if pair programming is a clear preference (The t-test result for question B5.1 "I prefer using pair programming technique over individual programming technique" did not produce a statistically significant result). This means they may still prefer individual programming although responses to other questions indicate a positive response to pair programming. Question B3.4: "I prefer working in a team to working alone", suggests a positive attitude towards collaboration and participation (key elements of pair programming) with which there was significant agreement. This challenges the results of previous studies which indicate that software developers prefer to work individually rather than in pair programming (Vanhanen, 2005; Williams et al., 2000). The responses to question D2.3 are inconclusive: the software developers seem to feel that, even though they might not have been adequately trained in pair programming, they seem to agree that they feel they have the skills and knowledge to be able to handle the different aspects that are required in pair programming. Thus, although the key question (B5.1) produced inconclusive results there are numerous indicators that

these developers support the use of pair programming; with some indicators suggesting they may value some aspects over individual programming.

5.8 Correlation results of major constructs in Theory of Planned Behaviour

The Pearson and Spearman tests were used to determine the correlation. The Pearson test is used for parametric data and Spearman is used for non-parametric data. The Spearman's test is used for the questions containing ordinal data, such as question A.1. The Pearson's test is used where the question involves scaled data, such as question A.2. These two tests are used to analyse the strength and direction (positive or negative) of relationships between two variables. If the results indicate a positive correlation, then it implies that as one variable increases the other will also increase. However, if there is a negative correlation, then as one variable increases then the other variable will decrease (Pallant, 2001).

A correlation test was performed with the four independent variables of the construct attitude against the factor of behavioural intent. The sub-themes of attitude are attitude towards change, learning, and collaboration/participation; attitude towards pair programming; and preference for pair programming versus individual programming. The results show that a positive correlation exists between the four independent variables and behavioural intention. The statistical results are:

- Attitude towards change and behavioural intention $r=.651$, $p=.005$ ($p<.05$)
- Attitude towards collaboration and participation and behavioural intention $r=.788$, $p=.000$ ($p<.005$)
- Attitude towards pair programming and behavioural intention $r=.641$, $p=.006$ ($p<.05$)
- Developers' preference for developing with pair versus individual programming and behavioural intention $r=.663$, $p=.004$ ($p<.05$)

This shows that the high scores of the independent variables correlate with high scores of behavioural intention. The software developers who agree or disagree with the factors of pair programming will also have the same attitude towards behavioural intention, which is the intention to use pair programming (See Appendix H: Correlation Statistical Results).

The next correlation test was performed using the responses to questions A.1 and A.2, dealing with the length of time the developers have been using pair programming and the extent to which they currently use pair programming. Another correlation test was performed with the sub-themes of the constructs: subjective norms and perceived behavioural control. The Spearman's test was used to determine correlation between the length of time using pair programming and the sub-themes of subjective norms and perceived behavioural control. The sub-themes of subjective norms are senior

staff influence and peer influence. The sub-themes of perceived behavioural control are resources and environment and skills/abilities. The results of the analysis show that positive correlations exist between the independent variable of skills/abilities within perceived behavioural control and question A.1: 'How long have you been using pair programming?', r=.491, p=.045 (p<0.05). This implies that the high scores of the independent variables (skills/abilities) are correlated with high scores in the use of pair programming length of time using pair programming.

The correlation between the attitude and question A.1 (How long have you been using pair programming?) was not significant; therefore, conclusions cannot be drawn about the nature of the relationship between question A.1 and subjective norms and perceived behavioural control, since there is no statistically significant relationship.

The results show a significant, positive correlation between the three independent variables of senior influence, resources/environmental and skills/abilities and question A.2 (I currently use pair programming in my projects). The results of the statistical analysis are as follows:

- Senior staff influence and question A.2 rho=.709, p=.001 (p< 005)
- Resource/environmental and question A.2 rho=.538, p=.026 (p<.05)
- Skills and abilities and question A.2 rho=.690, p=.002 (p<.05)

(See Appendix H: Correlation Statistical Results)

The high values of the independent variables (senior staff influence, resource/environmental factors and skills/abilities) correlate with the high values of the actual use of pair programming. Previous studies indicate that, with the right resources, people and tools, pair programming can be easily used (Cockburn & Williams, 2000). In addition, with the support of senior staff, other developers feel more at ease knowing that use of the programming technique is acceptable (Asaria et al., 2014).

The results of the analysis of the correlation between the constructs and question A.2 (I currently use pair programming in my projects) shows positive correlation for all three constructs. The results of the statistical analysis are as follows:

- Attitude and question A.2 (I currently use pair programming in my projects) rho=.744, p=.001 (p<.005)
- Subjective norm and question A.2 (I currently use pair programming in my projects) rho=.632, p=.007 (p<.05)
- Perceived behavioural control and question A.2 (I currently use pair programming in my projects) rho=.792, p=.000 (p<.005)

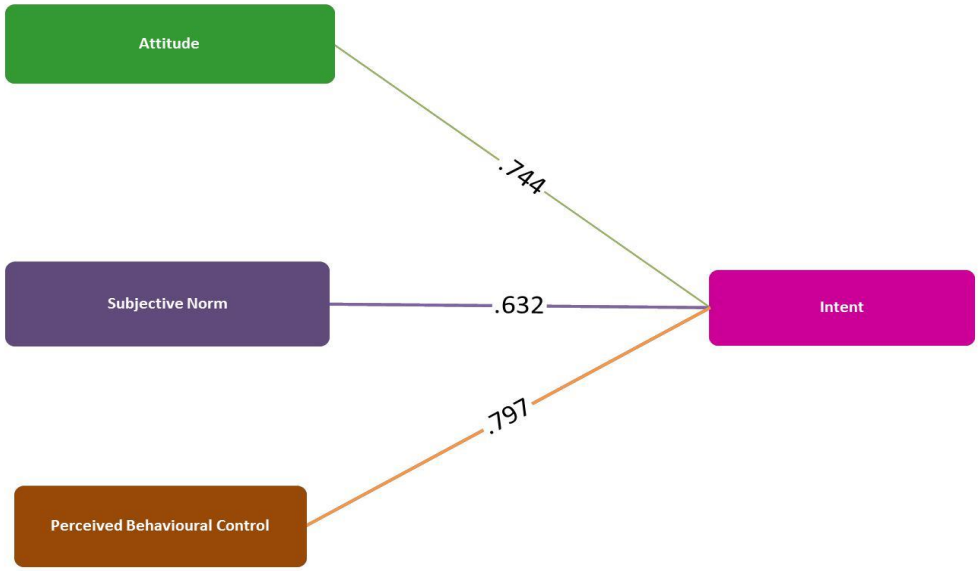
The results show that the use of pair programming is directly impacted by attitude, subjective norms and perceived behavioural control.

(See Appendix H: Correlation Statistical Results)

Summary of Correlation Results

The below present a summary of correlation tests per conceptual framework construct namely attitude, subjective norm and perceived behavioural control. The two dependent variables of the conceptual framework are behaviour and intent. The construct "behavioural intent" of the conceptual framework is used to represent the dependent variable 'intent' and the question A.2 "I currently use pair programming in my projects" is used to represent the dependent variable 'behaviour'. The dashed connection lines in the figures below represents statistically non-significant results and the solid connection lines represent significant results.

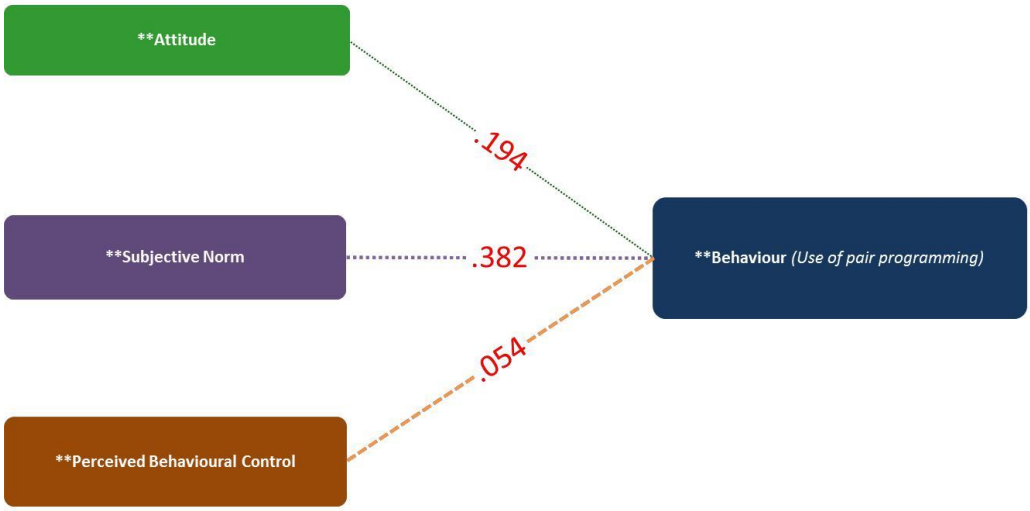
Figure 16: Correlation results between conceptual framework constructs and intent



The results in the above figure depicts that the construct attitude (rho=.744) and perceived behavioural control (rho=.797) has a greater correlation with intent. However subjective norm (rho=.632) also have a significant positive correlation with intent. Therefore, with a positive attitude, a pair-programming supportive company culture, and sufficient resources/environmental factors and skill/abilities the software developers are more likely to use pair programming.

and peerinfluence doeshave an impact ontheuse

Figure 17:Correlation results between conceptual framework constructs and behaviour



The results from the above figure depicts that the constructs attitude (rho=.194), subjective norm (rho-.382) and perceived behavioural control (rho=.054) has no significant correlation with behaviour. This could be because the measurement for behaviour is a historical value (how long it has been used) rather than a measure of actual future use. Future use, as a result of behavioural intent, cannot be measured unless a longitudinal study is undertaken.

This study makes use of the independent variables of the conceptual framework construct namely: attitude, subjective norm and perceived behavioural control. These independent variables therefore are tests for correlation again the dependant variables of intent and behaviour.

5.9 Conclusion

The results discussed above were obtained from the quantitative instrument, the questionnaire. The following was established:

The research question, “How do attitudes correlate to software developers’ use of pair programming within an agile software development methodology?” yielded significant results. This was broken down into the sub-themes of attitude towards pair programming; attitude towards collaboration and participation; attitude towards learning; attitude towards change; and developers’ preference for pair versus individual programming. Change is not a prohibiting factor that will affect the attitude of software developers using pair programming; instead the software developers are more accepting of change. In an agile technological environment changes are common, which increases the need to learn. This company also exhibits a positive attitude towards learning and is therefore able to cope with the dynamics of using pair programming. Due to the frequent partnering involved in pair

programming, the ability to work in a team and to work well with others is vital for success. This was positively noted in this company. The software developers adopt a positive attitude when working in a team and are able to work well with others and find motivation in teamwork. The benefits of knowledge sharing, quality work, improving self-esteem and improving productivity are regarded favourably. The developers favour working in a team rather than working alone, but disagreed significantly with the statement that pair programming made them feel more productive than programming alone.

The research question, 'How do subjective norms correlate to software developers' use of pair programming within an agile software development methodology?' also yielded significant results. This factor was broken down into two sub-themes: senior staff influence and peer influence. The results of the analysis of senior staff and peer influence were significant and in agreement. They suggest that the senior staff are supportive of staff growth and motivates the use of pair programming within the company. Peers are also in favour of using pair programming due to the benefits it provides. However, peers did note that with pair programming there are potential personality factors that would need to be considered to improve the use of pair programming, for instance, ego and issues of dominance.

The results of the analysis of the responses to the research question, 'How does perceived behavioural control correlate to software developers' use of pair programming within an agile software development methodology?' were significant. This factor was broken down into two factors namely: resource/environmental factors and skill/abilities. The results of the analysis of resource/environmental factors were significant and in agreement, indicating that this company provides the right, and sufficient, resources and maintains a good environment in which to use pair programming. The results of the skill/abilities analysis were significant and in agreement. The software developers feel they have adequate skills/abilities to use pair programming, even though the results of the analysis of formal training for pair programming was inconclusive.

The results of the analysis of the responses to the research question, 'How does intention correlate to software developers' use of pair programming within an agile software development methodology?' were significant. The results were also in agreement and show that the use of pair programming is promoted for current and future use. The results indicate that developers in this company are significantly in agreement regarding the use of pair programming and continue use of it. The behavioural intent of using pair programming currently, and increasing the use thereof, suggests the likelihood of using pair programming in future software development. There is a recurrent concern about pair mix, with respondents commenting on negative factors of dominance, ego, etc. It was noted that the optimal pair programming was between a senior and a junior programmer.

The correlation between the sub-factors of attitude and behavioural intention was significantly positive. There is positive correlation between the three constructs of attitude, subjective norms and perceived behavioural control and the intent i.e. behavioural intent conceptual framework construct". There was no significant correlation between the conceptual framework constructs and A.2 "I currently use pair programming in my projects" which relates to actual use of pair programming.

Chapter 6: Qualitative Data Analysis

6.1 Introduction

The data was obtained for the qualitative research by interviewing ten of the 17 participating software developers. The company limited the number of interviews to ten in order to minimise interruptions to production time; particularly since each interview took approximately one hour. This was confirmed in a telephonic conversation with the Human Resource representative on behalf of the director of the company. The interviews were categorised according to work level: senior staff (managers, team leaders or senior software developers) and developers (junior, intern or graduate software developers). Five participants from each level were interviewed (refer to table below that provides a list of participants' work positions)

This chapter will present the data obtained by using the qualitative research instrument as well as the results of the analysis of this data. The qualitative data was analysed using the NVivo Version 11 tool. The presentation of the results will follow a similar layout to that used in Chapter 5 (quantitative data) to maintain consistency. The results will be categorised by themes based on the conceptual framework constructs: attitude, subjective norm, perceived behavioural control and behavioural intention. The data obtained in the questionnaires (discussed in Chapter 5) was intended to provide a broad understanding which would be investigated in more detail in the interviews. Therefore, there may be references made to the results from the quantitative survey.

Table 4: Summary of participants' job positions

Participant	Job position
Participant 1	Junior Developer
Participant 2	Technical Manager
Participant 3	Intermediate Developer
Participant 4	Senior Developer
Participant 5	Senior Developer
Participant 6	Intermediate Developer
Participant 7	Junior Developer
Participant 8	Junior Developer
Participant 9	Technical Manager
Participant 10	Technical Manager

6.2 Overview of Qualitative Data

The qualitative results obtained from senior staff showed that they were positive about the use and benefits of pair programming. Pair programming is largely promoted within the company by senior staff, especially in order to mentor less experienced staff; particularly since pair programming is believed to increase sharing of knowledge and learning which helps to upskill less experienced staff and boost their confidence levels by working side by side with someone more senior. However, senior staff noted that less experienced staff can feel intimidated and are reluctant to voice their opinions. The software developers feel that it is challenging to pair program for many hours, as it becomes tiring due to the constant interaction. Developers at both work levels felt that pair programming is useful in improving their knowledge. However, it was noted that incompatible personalities could be problematic in pair programming, leading to issues with ego and criticism.

The junior and senior staff both commented on pair programming improving the quality of their code as it has two heads working on a task. The developers also felt that they had the right resources to support the use of pair programming. The developers did not discuss the agile approach much as they felt pair programming is a practice that is not reliant on a person working within only a specific methodology but rather that it can be used in a range of programming environments. However, they did indicate that 'agile' does support the use of pair programming: the principles of agile promote team work, communication and collaboration which are core dynamics of pair programming and this helps to encourage the appropriate mind-set. The senior and less experienced developers indicated that some of their internal processes improved the use of pair programming; for instance, 'focus times', which is an allocated period of time when developers avoid interruptions and work individually. They also conducted 'delivery practice', which was an allocated time that provided learning opportunities for all developers, as a group. These were a few tweaks in the process, instituted by the company, to improve the use of pair programming.

The results obtained from the interviews will be explained in more detail in the following sections. Sections 6.3 to 6.7 will discuss the results relating to the research questions dealing with the constructs of the conceptual framework. The results will be discussed at the sub-theme level, per construct.

6.3 Use of Pair Programming

The overall use of pair programming is extensively adopted in the company. In addition, it was indicated in the quantitative results that 76.5% of the software developers have been using pair programming for more than one year. The software developers (both senior and junior staff) are actively using pair programming in current projects. The senior software developers commented that "we definitely use pair programming for the typical advantages of quality and sharing but probably

the main place that I have seen it used is in terms of mentoring" (Participant 9); "as a training tool so I sit with the junior guys and we pair for short periods of time to get them to grips with concepts or pieces of the system" (Participant 5). Similarly, the less experienced staff noted that "when we struggled with something that we were stuck on, or something that we just want someone to look at a piece of code, then we pair up and it is always nice to get a second opinion on a code" (Participant 1) or "for learning purposes" (Participant 7).

6.4 Attitude

This section will explain the results obtained regarding the conceptual framework construct 'Attitude', which is linked to the research question: 'How do attitudes correlate to software developers' use of pair programming within an agile software development methodology?' This is broken down into the variables: attitude towards change; collaboration and participation; learning; pair programming and developers' preference for pair programming versus individual programming. The results are from the qualitative research instruments and provide the corresponding interpretations.

6.4.1 Attitude towards change

At an individual level, change can be easy to deal with if it does not affect the usual way of work. For example: "If … the changes are in alignment with helping me do my job, then that is easy to adjust to. If the changes are counter to my value system, or make it difficult to do my job, then that is difficult to adjust to" (Participant 6). However, irrespective of the type of change, the company "are extremely responsive to change mainly due to the methodology (agile) being part of applying learning and continuous improvements" (Participant 3). Some of the ways the developers describe their response to change are that they are "adaptive, respond quickly to change" (Participant 9). The software developers believe change is important to their roles as they are in a technological environment where change is constant "because programming, technologies, frameworks are changing so fast, if you do not keep up with the change then you are not really going to stay relevant; so change has to be embraced and stay relevant" (Participant 6).

This highlights one of the reasons agile developed – that technological environments in the 21st century are exposed to a rapidly evolving environment that bring a lot of changes (Balbes, 2014). Therefore, the agile methodology fosters the ability to adapt. In addition, since pair programming is an XP practice, it promotes incremental planning, which assists with accommodating and coping with change (Lindstrom & Jeffries, 2004). In addition, XP strongly embodies the principles of agile; particularly communication which enables feedback. This is essential in order to overcome change, especially when change is often present. The rapid communication provides a shared understanding and interaction between developers which leads to feedback. Therefore, risks can be mitigated earlier by questions being raised during continual communication (Konovalov & Misslinger, 2006).

Asaria et al. (2014, p. 74) indicates that "organisational change challenges the way things are done and as a result individuals experience uncertainty and start having fears of the potential failures in coping". Therefore, the level of success experienced in coping with the change will influence the attitude towards change. The participants mostly expressed a positive attitude towards changes and were highly responsive to changes. This suggests that the software developers expect change to be present to some extent, since a technological environment is constantly changing. If changes hinder the usual way of working, it may be difficult to adjust, and that may cause some discomfort or challenges. This is important as this builds a workplace that embraces 'agility'; one that supports environmental, technological or even method changes (Asaria et al., 2014).

As noted by Participant 4, some of the ways of coping with change is to "assess what the cost of that change is, to make sure that everybody is aware of the cost of those changes" (Participant 4); or "conduct an experiment…do some hypothesis; how you are going to conduct it. And you come back and do the evaluation afterwards with regard to whether the experiment was successful or not" (Participant 10). The teams hold discussions and reach an understanding of how each individual feels about things that work or do not work or can be improved. It is noted that the use of agile allows perspectives that "comes from within the team. It has got more power and also more power to enact and enable that change" (Participant 9).

6.4.2 Attitude towards learning

The software developers strongly support learning as "if you are in a dynamic environment, then it is important to keep up with the change. If you are a developer who is comfortable… where they are … and you are not required to learn much, then you do not have to... I would say, as the rule of thumb… In general, if you want to be a good software developer you need to learn. You need to continually develop yourself to do your work well" (Participant 6). It was also noted that pair programming encourages learning as "you get to see how they (software developers) think and solve problems, when you are pairing with them. You are sort of working on different things. You learn a lot of stuff from them" and "you share your knowledge faster…you know… get people up to speed" (Participant 8). In pair programming the software developers are constantly sharing ideas, brain storming and negotiating on a piece of work. This inherently transfers knowledge and improves on knowledge sharing, thus improving on learning (Williams, 2010).

Agile is inherently about embracing changes and continually adapting to the evolving needs. Therefore the software developers expect learning, especially by being in a technological and agile environment (Boehm, 2006). Therefore, wanting to learn and being able to learn is something they achieve for themselves (a self-imposed responsibility), rather than something initiated solely by the company.

The company encourages the software developers to attend conferences and community events. The formal training that carries a cost generally requires some motivation, especially if it also requires travelling costs. However, there is a weekly learning session called 'delivery practice' (this is further discussed in section 6.5.2) that is held internally. This session allows software developers at all work levels to listen, understand and share ideas in a team forum. This empowers learning at all work levels and brings fresh ideas to the way of working. These internally held learning workshops are advantageous to the company by promoting learning in a space in which the software developers are comfortable and utilising internal knowledge to empower others. The positive attitude of software developers towards learning can improve the company's agility: being able to adapt and create new knowledge and transferring or applying knowledge; which inevitably creates a multifunctional team. (Asaria et al., 2014).

6.4.3 Attitude towards collaboration and participation

One individual noted that "one aspect of agile is team collaboration" (Participant 2) and with collaboration and participation "there is no rock stars or hierarchy or silos within a team, so it definitely improves relationships" (Participant 4). This suggests that using agile as a methodology supports and promotes team behaviour. This echoes the agile manifesto, especially 'Individuals and interaction *over* process and tools' and 'Customer collaboration *over* contract negotiation' (Lindstrom & Jeffries, 2004), which means that agile empowers an interactive, collaborative environment. Therefore, agile assists in bringing individuals together to speak, share ideas and thoughts and collaborate continually to ensure that the right thing is delivered and there is a common shared understanding (Abrahamsson et al., 2017).

The factors of collaboration and participation sub-themes of the conceptual framework are used to integrate agile into the questions of the survey and interviews. Questions in the interviews highlighted that the software developers see agile and the principles of agile as an important factor in using pair programming. The software developers have indicated that using agile has provided "structure whereby you have got a plan in mind. It is not just you throwing things all over the place" (Participant 1). Agile as a methodology has assisted in the use of pair programming due to the dynamics agile provides such as "…you get feedback sooner" (Participant 2). "We agree with the agile principles, so for us agile is a set of principles mainly focused on feedback and the way you collaborate with the customers" (Participant 9); and "agile is team collaboration" (Participant 2). Since agile is based on continual improvements "You try and deliver small features along the way. You can be kind of harping back and forth between things when you could be continually moving forward" (Participant 4). "Agile methodologies can help you to be more productive…..it facilitates pair productivity as the more we produce in increments the better we feel about ourselves" (Participant 7). This is "unlike waterfall approaches. You are not six months in working on a massive thousand page spec. You are

working on smaller user stories and you are delivering continuously to the client, which means those changes that feedback is smaller" (Participant 6).One of the benefits noted when using pair programming was that "in a team people have different opinions so we get to see the pros and cons of those opinions and sometimes they debate and you learn more from their debate as well" (Participant 7). Another benefit of collaboration and participation is that the "relationship is positively impacted because you have knowledge sharing that generally happens, so you generally learn a lot as you share ideas or different ways of doing things" (Participant 7). "You also become more excited with what you are doing because you have more people you are sharing it that has a collective ownership of code with the person you wrote it with. Generally, while human beings are intrinsically social, so you are able to work and be social at the same time so you generally have a better outcome it promotes better team work" (Participant 6).

It was also noted in the survey within additional comments section, there two respondents who voluntary indicated the following: "Pair programming's effectiveness depends on the people involved in the pair. Its benefits will be hindered if one member dominates, rejecting the ideas of the other. Personalities also influence the pair programming effectiveness. Pair programming requires egos to be put aside and earnestly considering the other individual's ideas". This implies that the pair mix, more specifically the personality mix of pairs for pair programming drastically impacts the usage. The incorrect mix may result in the developers experiencing "dominance, rejected ideas or ego issues". It was also noted by a junior developer that "they might feel overpowered because now personality sometimes comes to the picture" (Participant 1). However, pair programming between seniors is more about discussion – brainstorming and developing ideas. "If it is a senior team, they will tell me to get lost. They will do it their way. They will happily disagree with me and it will be much more a debate about how things are going to get done. If it is a very junior team then there will be much less room for debate" (Participant 2). All individuals have different personality types, skill levels, learning styles, programming self-esteem and work ethics. These differences may become an issue or cause conflict with another person. Therefore, in pair programming, it's important to ensure that pair compatibility is considered. The smaller the individual differences, the more compatible pairs are; which will reduce the conflicts and make pair programming more productive (Williams, 2006).

The participation of both the individuals during pair programming is crucial as disengagement negatively impacts the use of pair programming: "… person driving the keyboard; I find them doing all the work while the other person sits and plays on their phone whatever… so there is a level of engagement that needs to happen with both people" (Participant 2). This suggests that, when using pair programming, both individuals in the pair need to be actively involved. If one person pulls back or becomes inactive, then this will influence the productivity of pair programming. Tsyganok (2016) notes that, as a beginner developer, one of the issues experienced with pair programming is 'session management', which are the interruptions during pair programming sessions. As a solution,

Tsyganok (2016) suggested that junior-senior pairs adopt a pattern that includes the experienced individual as the leader and the less experienced individual as the adopter/learner. A "simple metaphor of a parent teaching their child to ride a bike might help. The parent tasked with teaching is expected to not only be a confident cyclist themselves, but to be also adequately patient and articulated to lead their child through learning experience. From the adopter's perspective, quality of pairing experience is heavily influenced by the emotional expertise of the leader. Continuing the cycling metaphor, the most effective teaching method involves a parent and a child working together as equal" (Tsyganok, 2016, p. 273).

The developers indicated that, when doing pair programming, they "code a lot quicker and more quality and there is also general assistance everywhere" (Participant 1). This is due to "two people looking at the code" (Participant 8). The developers have noted that with pair programming, "every bit of code that gets written … at least two pairs of eyes have gone through this…whatever you have worked on, that piece is done according to more than one person's understanding within the team....and some of them will go through it and have a look that it is actually working and making sure that they have actually tested the thing" (Participant 9). This suggests that team initiative and owning the code when pair programming motivates the need to review code, hence improving the quality. For example "…pair programmed is usually higher quality code because you have more than one person looking at the code and two heads are usually better than one …And usually the other person serves as a safety net so you are able to pick up bugs before you release your code…..Pair programming, we solve the problem a lot quicker and we solve it better so we get cleaner code; we get more performance code and it usually has less bugs" (Participant 9). This concurs with previous studies that indicate that, since pair programming has two minds involved continually, they are able to collaborate and resolve issues earlier, thereby producing higher quality work (Balijepally et al., 2009; Begel & Nagappan, 2008).

The junior developers have indicated that "…it was more productive because the extrovert goes on to teach mode and when he explains, he explains till he gets it into your head" (Participant 7). This implies that an extrovert tends to speak more, explaining things more than would an introvert. It was noted that introverts do not voice their ideas as easily, However, this could also apply to extroverts, if they are "someone fragile and someone who feels will be criticised afterwards…. and they will not be willing to ask questions" (Participant 7).

In addition to the personality conflicts are the 'pair-related' factors that disturb the use of pair programming: An example is the break in pair programming with conversations, like "busy working on a solution and your partner all of a sudden…. 'I watched this last night it was so good'… and that immediately breaks your train of thought and you kind of lose your track of where you were" (Participant 10). Previous studies indicate that session management is crucial in pair programming, so

the session of pair programming must be made as comfortable as possible and with no distractions, so that the best productivity of the session can be achieved (Tsyganok, 2016). This impacts the use of pair programming as the software developers feel "you absolutely cannot work if you keep on getting interrupted and disturbed and things like that…so during that focus time we encourage people… Close your email. Put your phone on silent. Try to minimise disruptions" (Participant 9).

To counteract the potential for disruption or distraction, the company has enforced a session called 'focus time'. This session occurs almost every day. Two hours or more per day are set aside for developers to focus only on programming individually with no interruption. The developers generally "plug in the headphones to minimise sound around them" (Participant 9). However, focus time also accommodates pair programming: "some people will be pairing around focus time so they discuss it but they are both sharing the same context, both working on the same thing" (Participant 9).

Asaria et al. (2014, p.75) indicates that an "agile workforce is also expected to effectively take part in any collaborative environment whether it's cross functional, collaborative ventures". This is associated with an attitude towards team work which portrays an "individual's willingness to work with others". The agile methodology strongly supports teamwork, which is reflected in the software developers having a positive attitude towards building a stronger team. However, with the benefits come some downfalls. It is noted that "There will be some level of disagreement and how do you resolve that? So you need to be cognisant if you are both trying to solve the same thing then you come up with a strategy… If you come up with different strategies you must be able to say…..this is how it goes… or talk it through. To me … without taking it as a personal attack, because you have different views on how to do things, it means that we both find a way that we are both happy with" (Participant 4). It was noted by a junior software developer that, through collaboration and participation, they "get to be confident, because some people are nervous you get to, (at) some level…, know the person better that you are working with" (Participant 7). This suggests that pair programming helps less experienced staff improve their skills and build a better relationship with their colleagues. As the software developers have a common goal, their disagreements can enhance the thinking and debates. A productive disagreement comes from being able to move past a disagreement to find a solution (Balijepally et al., 2009).

6.4.4 Attitude towards pair programming

The attitude towards pair programming is in general positive. The software developers have indicated that pair programming "positively influences relationships, because you have knowledge sharing that generally happens, so you learn a lot as you share ideas or different ways of doing things" (Participant 6) and with having more people involved "you get a buddy" (Participant 8). Pair programming as an approach normally involves a transferring of knowledge as 'two-minds' are involved' (Begel & Nagappan, 2008). This is because of the dynamics of pair programming, which includes constant

conversations, brainstorming, sharing of ideas that allow for early feedback, and understanding. This then empowers individuals within the pair to be knowledgeable. Due to the constant sharing of knowledge, there is no dependence on one person "you do not get silos, like someone off sick or gets here by company or gets here by bus. You have not got a silo – that is the only person that knows the code; and also you end up getting a code that is easier to understand because a code has two consumers" (Participant 10). Therefore "(with) pair programming you are increasing the human readability and comprehensibility ….especially if you are switching pairs within a team. The team will end up getting into an aspect of synergy around how a solution comes together … and styles and things like that. So that you do not get areas of the code so kind of far different to the code base. Your code will have more of a familiarity across" (Participant 9). Hence, pair programming is beneficial in upskilling those with less experience. It also assists in resolving errors or issues earlier and reduces a one-person dependence as another person shares the understanding (Kim et al., 2010).

It was noted by a senior staff member that "if you do not pair, you are kind of sitting in a bubble… I think it is also, if I come in and sit with a junior then I am learning. I help other people to understand and I think for the juniors they have, like a safety belt, with the seniors. They do not feel like they are going to fall over" (Participant 4). This suggests that pair programming positively improves team relationships and assists the development of juniors by upskilling and building their confidence. It was noted by (Tsyganok, 2016, p. 271) "When I was a junior developer and in a room with lots of different stakeholders, you sometimes do not raise your fears because you are not a hundred percent confident". This suggests that the senior software developers of this company commonly adopt pair programming as a training-tool for those less experienced. This is a popular approach in pair programming and is sometimes deliberately done, whereby a junior is paired with a senior so that they can leverage off each other simultaneously. The junior developers expect to improve their skills and abilities, whereas the senior developer is expected to learn how to interact and grasp the fresh new ideas. However, less experienced software developers are reluctant to voice their opinions. This was also noted within this chapter in the section 'attitude towards collaboration and participation'. This could be an issue that impacts the use and effectiveness of pair programming, especially for less experienced software developers.

Differences in personalities can create conflict when using pair programming. In teams that do not always go well together, there is a personality conflict or you have a different social dynamic within the team for instance: "…too many people with different opinions and they struggle to agree with things…I think if you can work things out with your team and you have good processes and you have people that are good team players then it is good" (Participant 5). However, the senior staff use it more as an approach to mentor and boost the confidence level of the juniors. Vanhanen (2005) indicates that it takes the right combination for a pair to be productive when using pair programming.

It was also noted that pair programming "improves quality of our codes, as pair programming involves code review…we spend a lot of time discussing our code and understanding how we can use it" (Participant 2). Due to the constant sharing of ideas, which enables the software developers to have a good understanding of the code being written, they are able to provide feedback often and earlier (Schmidt et al., 2014). Fu et al. (2017) notes that, in pair programming, more defects in the code were realised earlier and fewer tests were required later.

A senior software developer noted that a perceived negative impact of using pair programming is "two people doing the same thing and you are achieving less" (Participant 9). But it is not the case, since "it's very much like construction….. so they are like…. I get two people to build one wall, will go faster than if I get two people to build two separate walls" (Participant 3). This company is very supportive of the use of pair programming. However, other companies are not, due to the negative perceptions. A company culture which supports employees and their needs will encourage the use of pair programming, as the software developers will view it as favourable to use pair programming; and even if there are issues that affect its use, the company is there to support their needs (Asaria et al., 2014).

6.4.5 Developers' preference for developing with pair programming versus individual programming

A software developer comments that pair programming is working as "a team …where as an individual you do some parts of the job. However, when you are working alone you get the whole job for yourself" (Participant 1). "There are benefits for both sides. When you are working as an individual you tend to struggle a lot…in a team things are done quicker" (Participants 7). It was also noted that with individual programming it can be "a very mentally taxing process … all the biases that you are subjected to as individuals, you are never going to get the perfect solution as a single person. But you have other people coming in to offset the biases and you end up with a better solution" (Participant 10). This suggests that with pair programming there is shared responsibility and two individuals assisting one another; whereas working individually means having to complete the task on one's own (Williams et al., 2000). It was noted by a junior software developer that pair programming is not beneficial in all circumstances: "if you don't know exactly what needs to be done, then it would be better for you trying to solve solution alone than working with pair programming" (Participant 8). The behaviour of the junior staff is due to their limited knowledge and they withdraw at times: "I will first research, because they have much more experience than I do, before I try to challenge. I do not want to make premature decisions, and once I have a better understanding, I will try to comment" (Participant 7). This suggests that when less experienced software developers are unsure of the details of the task, they prefer working through this on their own. It could be that, if they do approach an experienced software developer about the task, they would want to know enough of the details in

order to engage in the discussion. In simpler tasks there is less need for pair programming. However, when the task is more difficult and takes longer time to resolve, then there is more of a need to seek help, and the use of using pair programming is deemed necessary (Williams et al., 2000).

Williams (2010) indicates that the dynamics of pair programming are vastly different to the traditional way of programming individually. It was stated by a senior developer that: "I think if you were to take everybody's time, the percentage when spent with pair programming and the percentage spent alone, I think the percentage programming alone would be higher" (Participant 5). This could be a single individual's perception, since it was not repeated by other participants and the software developers spend some of their time programming alone as part of their internal practice called 'focus times'. The less experienced developers feel "reluctant" (Participant 1) to voice their opinions to more experienced developers. This could suggest 'ego' issues, and the less experienced developers want to uphold a socially acceptable image.

6.5 Subjective Norms

This section discusses the results obtained regarding the conceptual framework construct 'Subjective Norms' which is linked to the research question: 'How do subjective norms correlate to software developers' use of pair programming within an agile software development methodology?' 'Subjective norms' include senior staff and peer influences. The information which is discussed and interpreted was collected by the qualitative research instrument (the interviews).

6.5.1 Senior staff influence

The influence of senior staff on other staff is a crucial factor that can either enable, or be a barrier to performing in a certain way (Asaria et al., 2014). The senior staff mostly noted that pair programming is the norm within the company. However, the software developers have freedom of choice in deciding which programming method they use.

The senior staff members at the company have been very supportive in sharing their knowledge with other developers, especially those at lower work levels. This was noted by a junior developer: "senior staff support, especially in helping me learn to develop my software skills" (Participant 1). In these cases, senior staff "prefers to challenge people to be better, so I am perpetually trying to take people out their comfort zones and show them that they can be better. If I had a choice of giving them an answer to a question they asked or asking them a questions leading to the answer, I would always favour leading them to the answer. So there is a strong mentorship rather than… 'oh, that is how you solve it or there is the link, go and look through the page'" (Participant 3). This implies that the senior staff adopts a supportive approach with other staff and also likes to be directly involved to improve their learning: "more hands on…… I do like to have debates and discussions…….and we will try and come up with new ideas rather than just issuing, and asking for help and being spoon fed the answers"

(Participant 3). The positivity and support of the senior staff influences the behaviour of employees. They become 'social referees', determining whether the staff is doing the right or wrong things and even if the wrong things are done they can be appropriately supported in guiding them towards the correct way (Asaria et al., 2014).

6.5.2 Peer influence

The software developers noted that the company has a strong teamwork culture "…outside my team … I would talk to other team leaders and other people in the company because those improvements will impact my team; because if I learn something I am going to bring it into my team. I am not going to wait…" (Participant 3). The developers "respect their (peers) views and opinion" (Participant 8). They tend to consult their peers, not just for upskilling, but to "help me improve personal growth" (Participant 8). The senior staff indicate that they are able to have "peer discussions over coffee and once again this comes back to open office and just listening to what other teams are talking about….so how other people solve problems. I definitely consider how I tackle my problems" (Participant 5). In addition, the peer relationship within the company is noted to be "supportive" (Participant 1) as they "can seek guidance" (Participant 6) and the colleagues, irrespective of work level, "do not boast their work and they are not afraid to share knowledge" (Participant 7).

The views and thoughts of peers' impact on the behaviour of individuals, as individuals associate themselves with the views of others around them, especially in an unfamiliar situation. In addition, individuals deliberately reduce efforts that do not conform to the usual way of working, to avoid peers being unhappy (Asaria et al., 2014).

6.6 Perceived Behavioural Control

This section discusses the results obtained regarding the conceptual framework construct 'Perceived Behavioural Control' which is linked to the research question: 'How does perceived behavioural control correlate to software developers' use of pair programming within an agile software development methodology?' This is made up of the variables resources/environmental factors and skills/abilities. The information which is discussed and interpreted was collected by the qualitative research instrument (the interviews).

6.6.1 Resource and Environmental factors

The software developers explained that their work environment is suitable to use pair programming as they have "two keyboards, two monitors, so here we are brilliantly set up for pair programming. It is something that we have done quite a bit over the last couple of years so it is almost engrained in the culture" (Participant 2). The "desks they are pretty much straight, but there is enough room for two people to sit in and it is straight so at the curved spot it is a spot for one person" (Participant 9) and (with) "the open plan office it is very easy to pull out a chair and use someone else's desk. All the PCs

have two very big monitors and there are always extra keyboards and mice available if you want to plug into one computer, so it makes pairing a lot easier" (Participant 5). In addition the "you are free to use whiteboard" (Participant 10), which is a tool that assists in brainstorming. A common reason for not wanting to do pair programming is not having the adequate space or unsuitable equipment (Tsyganok, 2016). Since the dynamics of pair programming are vastly different from traditional, individual programming, the need for a suitable environment is vital. If you were to just use the traditional setting for pair programming it would not be effective and may even prohibit the use of pair programming (Williams et al., 2000). This company has the correct, and sufficient, resources and environment, which makes pair programming easy and less challenging to use (Tsyganok, 2016).

The developers indicate that, if additional resources are required for them to function, they "just tell them that I need the thing … ask and then it normally happens" (Participant 3). However, there are times "you may need to motivate, of course, what is the reason?" (Participant 9); but mostly a verbal request is sufficient. This suggests that the company considers the requests made by employees, especially those that impact the way they work; and by fulfilling the requests, they build a relationship of trust between the management and employees, developing a collaborative and supportive company culture (Asaria et al., 2014).

The developers indicated that at times the lengthy conversations conducted during pair programming become disturbing, especially if an individual speaks loudly; "one of my team members is quite loud. He is just talking normally and will cut across everything…." (Participant 4). Therefore the developers suggest having "some partitioning" (Participant 4) or "an enclosed space – just a place where two people can sit" (Participant 9). Previous studies indicate that a key factor in using pair programming successfully is having an environment that supports it. Since the dynamics of pair programming differ vastly to traditional programming, with two software developers working at one desk with one computer, it is necessary to accommodate the physical and resource needs of pair programming in order to use it easily (Williams, 2010). This requires resources that provide a desk space big enough for two people to work and an environment that can allow conversation. If the environment does not suit the practice, then negativity around the practice and its lack of adoption can be understood (Asaria et al., 2014).

Since most of the software developers are consultants, they sometimes work at geographically dispersed locations using technology to assist with the communication: "sometimes I consult clients in other areas and we use Skype to call each other and also sometimes pair program by shadowing other person's screen" (Participant 2). Pair programming is sometimes used in these instances through Skype. This has not been much of an issue, as the software developers have noted that they can share their screens and discuss code. However, at times there are network connection issues and in these instances pair programming is used for a very short period of time. This type of pair programming is

commonly referred to as ‘distributed pair programming’(Williams, 2010). This company practice supports the findings of a previous study that noted that distributed pair programming can be practiced with the support of a cross-workspace of visual, manual and audio channels. As this company makes use of Skype, which contains visual, audio and manual tools, the software developers are still able to pursue pair programming activities (Williams, 2010).

6.6.2 Skills and Abilities

The software developers have indicated that their soft skills would need to be improved when using pair programming. “I think people give it their best when they are more accepting and more open about their views and ideas. So less detachment from your work, so that you can work with somebody else and not feel offended if they do not like the way that you want to do it. You should try to solve the problem in the best way possible so I think a lot of people need to learn how to put ego out the way” (Participant 4). When using pair programming, “you have to see the problem as the focus, you have to almost see yourself as the team trying to solve this problem so you are both trying to achieve the same goal so whatever you need to achieve to get to that goal” (Participant 6). This implies that the software developers are able to function within the dynamics of pair programming. However, some negative aspects such as attachment to code and not being able to accept criticism are experienced and can affect the use of pair programming.

The software developers indicate that their communication skill would need to improve when using pair programming, since one needs to “communicate your thoughts and ideas….communication is a big thing” (Participant 1). “As a software developer you sometimes like to enclose yourself in a box and you definitely need to open up a bit more and be interactive, so that is a certain skill” (Participant 1). The need to improve communication skills were also noted by a junior develop: “… speak up more often because I am naturally quiet so I just need to speak up more often and voice my opinions” (Participant 7). As part of communicating they would also like to improve their listening skills: “you need to be able to put your thoughts into it” (Participant 1). Listening also helps: “to get constructive feedback you have to be able to listen well because a lot of people are not effective listeners; but if you do not listen to your partner when you are pair programming, you are never going to arrive at a consensus on what to do because you have not heard their viewpoints; so you have to listen well. You have to give constructive criticism. You have to be able to know when to listen and when to talk” (Participant 6).

The software developers attempt to improve their skills and abilities in-house by conducting ‘delivery practice’. Delivery practice is held every Friday at the company’s site. This involves all the software developers at all work levels. The session is approximately two hours long in the afternoon. The session can “take different forms, it depends on what we are trying to learn at that stage, so that can either be going off and tackling a problem with pair programming or it can be a mob session where

we have one screen and we are trying to solve the problem altogether" (Participant 5). The intention of this session is "upskilling the people that work here and that need the skills…. Our delivery practice discovers a lot of those skills. We have a lot of those discussions with the entire development team. They are either watching a video on programming or having discussions around software. They might be going over a book where the junior guys discuss in detail: 'Do you put code comments? Do you do pair programming or don't?' We spend a lot of time around those things." (Participant 2). However, if attendance at formal courses is required, the company is willing to fund this but will require motivation as the budget of the company will be considered. "There are finances to further education but we also do deliverable practice internally. We are all very open to sharing our knowledge and sharing what we have learned" (Participant 2). Pair negotiation and brainstorming are important components of the success of pair programming. The internal practice of delivery practice allows for brainstorming and active negotiation which does not only allow software developers to realise and discover new techniques or approaches but also to find solutions efficiently without causing frustration and decreasing productivity (Williams, 2010).

6.7 Behavioural Intention

The factor 'behavioural intention' is the main focus of this study, which is linked to the research question '*How does intention correlate to software developers' use of pair programming within an agile software development methodology?'* The behavioural intention is the use of pair programming, which is the overarching factor in the sub-themes within the conceptual framework (attitudes, subjective norms and perceived behavioural control). The information which is discussed and interpreted was collected by the qualitative research instrument (the interviews).

6.7.1 Intent

The software developers believe strongly that pair programming should currently be used more and should be promoted for future use. However, as mentioned in the above sections, pair programming is commonly used as an approach for upskilling or training less experienced staff. Pair programming involves a lot of knowledge sharing and the pairing tends to be seen as a "safety belt" (Participant 4) for less experienced staff. "Pair programming should be promoted for current and future use. You might not get the full value if you try and introduce pair programming full time as a preferred technique ….. It might be something that will work well for a three month project but I think it should be used appropriately to increase motivation, to share knowledge if it is needed to work on programs where you foresee something getting done" (Participant 2). It was also indicated that "with pair programming, the more complex the task, less experienced developers find themselves 'stuck in rabbit holes'". However, senior staff members given less complex tasks find themselves "bored" (Participant 9). This suggests that pair programming may be more or less beneficial at different work levels; since it is perceived that, on simpler tasks, senior software developers find pair programming less useful.

This supports the practice of adopting the use more as a tool to assist less experienced staff. A previous study supports this conclusion when it notes that, pair programming is used less "on simple coding of a project. However, they mostly pair program when code is of average complexity modules and if the situation, such as time conflicts, dictates – though most immediately feel notably uncomfortable and more error prone" (Williams et al., 2000, p. 5). This is because complex tasks require much prototyping and brainstorming, which becomes a tiring and difficult exercise for an individual to conduct (Williams et al., 2000).

The software developers favour pair programming as "you are creating an environment that encourages collaboration so you are free to write code; you are free to have conversations; You are free to go (to the) whiteboard if you do not understand it… You just stick to a creative environment. It is more conducive to the view that it is not just an individual plugged into the assembly line. I think it is a team working together to achieve a common goal" (Participant 10). Pair programming "just encourages the right type of mentality around software. It encourages knowledge transfer" (Participant 10). "Two people can actually be more productive than when they are apart and have different skill levels. The juniors can learn from the seniors and the seniors get an opportunity to mentor" (Participant 4).

However, pair programming can "become(s) a burden" (Participant 2). "It requires more energy from you, simply because you have got to constantly engage with the other person and you have to make your actions quite explicit and you have to constantly communicate with the other person because you are pairing the whole day" (Participant 2). Also, pair programming continually involves interaction and problem solving with another individual, the process becomes tiring; which is a factor that affects its adoption (Williams, 2010). A suggestion to improve the use of pair programming is to "take frequent breaks so that it is not too draining and doesn't require much energy. If you both agree to practice like some sort of technique… but you have certain time management techniques… like your focus time in between? So that might be effective in a pair programming scenario so that you do not have that sort of burnout if you are doing it for too long" (Participant 6).

6.8 Conclusion

Pair programming is used a lot within this company and is considered more of the norm than the exception. The pairing of software developers commonly involves a junior with a senior. The overall attitude towards pair programming is positive, notwithstanding some issues. The software developers have indicated that the benefits of using pair programming range from producing higher quality code, because you have more than one person looking at the code, and being able to pick up bugs before you release your code. In addition pair programming improves sharing of knowledge and communication skills. As an example, if a person that is shy is involved in pair programming they have to talk, as part of the pair programming process, so they will begin to form a relationship with

their colleagues. Pair programming also removes the biases that you are subjected to as individuals as having other people coming in to offset the biases and provide additional input helps you develop a better solution.

However, there are challenges experienced in pair programming. It was noted several times that the junior developers are reluctant to use pair programming as they feel too intimidated to voice their opinions due to the lack of experience. A major concern with pair programming is the personality conflicts that make individuals feel frustrated or reluctant to interact.

The developers indicate that pair programming is used mostly as training and mentoring tool that pairs up junior and senior developers. Pair programming is also used at a project level to encourage communication and collaboration.

The company makes provision for the important resources and tools needed to use pair programming and adopts a supportive culture by considering any further requests for resources The company also makes use of many internal practices to improve the practice of pair programming, like delivery practice to encourage and empower learning and improving of skills and abilities; as well as implementing focus time which is used to minimise disruptions and allow developers to focus on a specific, individual task.

However, pair programming does become tiring and requires a lot of energy due to the constant interaction. Factors such as regular breaks or switching of roles need to be considered to enhance the synergy in pair programming. Overall, the company intends to continue the use of pair programming both currently and in the future. Pair programming emerges strongly as more of an approach to upskill less experienced staff within this company. The next chapter will reflect on the findings of the quantitative results in conjunction with the results of this chapter, to provide concluding ideas and practical implications for practitioners.

Chapter 7: Conclusions and Implications for Practitioners

7.1 Introduction

This chapter will collate the findings from the quantitative data noted in Chapter 5 and the qualitative data noted in Chapter 6. The findings will be aligned to the research questions of the study as noted in Chapter (section 1.3). The results from both quantitative and qualitative were categorised by the constructs of the conceptual framework namely: attitudes, subjective norm, perceived behavioural control and behavioural intention. Each construct of the conceptual framework contains sub themes; therefore the data was collected at a sub theme level and the discussions are retained at the sub theme level as per the conceptual framework constructs. The intention of this chapter is to draw a conclusion for each research question. The conclusion will cover the theoretical implications of both the qualitative and quantitative results. This chapter will also provide implications for practitioners with relation to the use of pair programming in the software development industry.

7.2 Overview of the Use of Pair Programming

The company that was used as part of this research contained a balance of work experienced software developers (i.e. senior and junior staff). However most of the responses were obtained from males with only one female sampled out of 17 individuals. Therefore the perceptions obtained in this study are more from male perspectives than females. The global representation of females in software industries has drastically been reducing since 1991 (Ashcraft et al., 2016). The major contributing factors of this phenomenon is cultural challenges that restrict females as, "though female leaders have the same technical challenges and are expected to produce the same kind of results as male leaders, there is often a cultural context that influences their approach and a different interpretation of their performance that ups the ante" (Klawe, Whitney, & Simard, 2009, p. 73).

It was also established that the quantitative results were significantly in agreement and the qualitative results were positive about the use of pair programming. The quantitative result indicates that 76.5 % of the software developers in the target company have been using pair programming for more than 1 year and the majority significantly agree that they still use pair programming in current projects. The results from the qualitative study depicts that pair programming is favourably adopted due to the benefits realised from the use after adoption of pair programming. The literature indicates that pair programming is the least adopted and least popular practice within agile methodologies (Doyle et al., 2014). Due to the results obtained in the quantitative study being different to the previous studies findings, the use of pair programming was not expected to be largely adopted in general, but this study makes deliberate use of a company that actively uses pair programming so use should be high. This was necessary in order to ascertain the factors that influence pair programming adoption.

Since the software developers in this company are experienced in the dynamics of pair programming they are able to relate to the practice and depict its more or less favourable aspects. The practice is well recognised for improved knowledge sharing. However since the practice is commonly used to pair experienced and less experienced software developers, the most common benefit is known as upskilling less experienced developers. This is advantageous to the company as they also have an even spread of work levels (i.e. senior and junior software developers). As Dillenbourg (1999) indicates, pair programming allows two individuals to converge as one which mediates a process of sharing knowledge and the experience thus becomes more useful for improving skills.

In addition the constant interaction amongst two-minds facilitates feedback which assists in gaining reviews earlier and mitigates future risks (Cockburn & Williams, 2000; Fu et al., 2017).

7.3 Attitude

The research question that is aligned to attitude is "*How do attitudes correlate to software developers' use of pair programming within an agile software development methodology?*" The sub themes of this construct are: attitude towards change, collaboration/participation, learning, pair programming and developer's preference for developing using pair programming versus pair programming.

Attitude towards change

The sections B1.1 – B1.3 in the quantitative results depict overall positive results with the data showing that the survey respondents indicated a significantly positive attitude towards change. The software developers indicate that they are happy to change the way they do programming if it results in better outputs despite the potential discomfort of changes. The software developers also indicated a positive attitude towards change within the qualitative results. They did suggest that a change that may directly affect their way of working is more difficult to adjust to. However since being in a technological environment that adopts agile methodology, change is expected.

Since the software developers are able to cope with change they are able to experience uncertainties and cope in new situations. The software developers are optimistic about change and perceive change as an opportunity rather than a threat (Asaria et al., 2014). This is important for an agile organisation as change is instilled within the methodology. The manifesto which reads "respond to change *over* following a plan", indicates that agile teams must be proactive to change rather than following plans (Lindstrom & Jeffries, 2004). This responsive characteristic is displayed in this company which also enhances its competitive advantage (Asaria et al., 2014; Boehm, 2006; Lindstrom & Jeffries, 2004).

Implications for practitioners: Attitude towards change

Pair programming exposes developers to changes which may be difficult to deal with (refer to Chapter 2.3). In order to deal with change the following is suggested: change needs to be introduced in intervals to allow software developers to ease into the change (Asaria et al., 2014). If the change is likely to have a high impact on their productivity then they need to be more carefully handled and possibly introduced in periods where it does not affect core operational time, so that productivity is minimally interrupted. The change would require inspirational leadership and understanding of senior staff so that the developers find their environment supportive if fears or mistakes are experienced (Asaria et al., 2014).

Attitude towards learning

The sections B2.1 – B2.3 both the quantitative and the overall qualitative results showed positive attitudes towards learning. The quantitative results indicate that the software developers were highly positive towards the willingness to learn about new programming tools or techniques. They are keen, optimistic and highly interested in learning new things about design issues in software development. They see learning as vital to fulfilling their role in a technological environment where there are so many changes and use it to improve sharing of knowledge which in turn assists with learning.

In addition, it was indicated in the qualitative data that the company also adopts an internal practice called ‘delivery practice’ which is a session that consists of all software developers meeting to share ideas, discuss and understand what is required (discussed more in detail in 6.5.2). The ‘delivery practice’ session assists with sharing of knowledge and learning throughout the company and at any work level. The company also displays a supportive culture to learning requests that require external learning.

Implications for practitioners: Attitude towards learning

The company adopts an internal approach called delivery practice. This practice is a good approach to improve the learning of all software developers by using internal skill sets within the company to advance training instead of investing in external training programmes. Since this practice makes use of internal skills, this is a cost effective approach. In addition, this practice can target the needs of the current or upcoming projects or specific learning desires of employees. However this may limit the more general or global software development knowledge that the developers have, and which they can attain by attending formal workshops or seminars. Leadership or senior staff need to remain aware of formal learning opportunities that can be beneficial to the company and help the software developers in gaining a wider perspective of the industry.

Attitude towards collaboration and participation

The reliability results from Cronbach Alpha depicts a value of 0.568 (i.e. < 0.7) for attitude towards collaboration and participation. This does indicate that this construct is below the reliability threshold and results should be treated with some caution. However this reliability value could be a result of the small sample size.

The sections B3.1 – B3.5 in the quantitative results depict significant agreement. The qualitative data were also positive however they also raised many concerns about things they experience when using pair programming. The software developers have indicated that the company is very agile as they strongly support "individual interaction *over* contract negotiation", which indicates that stakeholders must be continuously communicating and ongoing collaboration is necessary to ensure good understanding and delivery of what is expected (Konovalov & Misslinger, 2006; Lindstrom & Jeffries, 2004; Schmidt et al., 2014). The quantitative results indicate that the software developers were significantly positive towards following the process defined and agreed upon. The team and the software developers feel motivated by other team members to participate in the development processes. In addition the frequent engagement when working in a team helps one build one's confidence and interpersonal skills, which also helps to improve individuals' knowledge, and empowers a shared understanding that assists in delivering code quicker with a higher quality (Cockburn & Williams, 2000; Williams, 2010).

However there are personality conflicts experienced that could lead to dominance of the interactions by specific team members. Extroverts that overpower other members can cause conflict and debates. Junior software developers may find themselves feeling reluctant to voice their opinions due to their lack of confidence and inexperience (Participant 7). By having pairs that are compatible reduces the conflicts and increases the comfort between team members (Williams, 2006). The interviews highlighted that there are other pair-related issues experienced when using pair programming, like disruption of work-focus with the conversations and even disengagement by an individual.

The literature suggests that one of the challenges when using pair programming is the disturbance from surrounding factors that impacts the use of pair programming (Williams, 2010). Since these distractions can break the focus in pair programming and become frustrating if it persists, this company adopts an internal practice called focus time, which dedicates time slots for software developers to work as individuals with minimal interruptions (Participant 7). The session empowers focus which inherently will improve productivity.

Implications for practitioners: Attitude towards collaboration and participation

Due to the dynamics of pair programming being so different to the traditional way of programming, the practice can be more beneficial if software developers are trained on how to work in teams and gain interpersonal skills for example guidance on how to deal with conflicts, especially when engaging with an individual that is deemed to have a personality which may lead to conflicts. There are additional measure that can be undertaken to resolve interpersonal conflicts such as: conducting regular retrospectives which are team meetings to discuss what works, what doesn't and how to improve; understanding the personalities involved so that an understanding of what individuals value personally as important or are crucial to them can be considered; and finding pairs that have aligned values.

Attitude towards pair programming

The overall quantitative and qualitative results were positive towards pair programming. The sections B4.1 – B4.4 and B4.6 - B4.9 in the quantitative results were significantly in agreement. The quantitative results also indicate that pair programming reduces the need for review/inspection of the code. Likewise, in the qualitative data the software developers indicate that by using pair programming there are admirable benefits realised that aligns to those mentioned in previous studies. These benefits include positively influenced staff relations and improved quality of work; as well as upskilling of staff and increased self-confidence, self-esteem and productivity (Balijepally et al., 2009; Begel & Nagappan, 2008; Fu et al., 2017).

However there was no significant result in terms of pair programming reducing the documentation required (question B4.5 "I feel pair programming reduces documentation") and the qualitative data did not discuss this aspect. In addition, the software developers noted a perception of pair programming that commonly exists and is a misinterpretation that negatively impacts its use, namely, that it is just having two people doing the same thing (Balijepally et al., 2009). However both the qualitative and quantitative data show that experiences in this company indicate this perception is not true. In addition, the company has the right hardware and tools to support the use of pair programming that makes it less difficult to use pair programming in its natural form i.e. having two individuals sitting at one desk working on one task on hand.

Implications for practitioners: Attitude towards pair programming

The software developers' attitude towards pair programming leans more towards being positive. By the leadership and senior staff continuously supporting the practice, other developers feel more at ease about adopting pair programming. In addition the company regularly assesses what aspects of pair programming are more beneficial and enjoyed. By doing this, the company can focus more on those

areas for improvement or common use so that the software developers will find the practice more appealing. Also, regular feedback sessions must be conducted to understand what does not work in pair programming to find the sore points that are meaningful to the software developers and need to be resolved (Tsyganok, 2016).

Developer's preference of developing with pair programming versus individual programming

The sections B5.3 – B5.4 in the quantitative results were significantly in agreement. In contrast to most of the responses, a t-test indicated respondents disagreed with the statement: B5.1 "I prefer using pair programming technique over individual programming technique". But this result should be treated as inconclusive because results were not statistically significant. Another question was statistically significant and indicated that respondents disagreed with the statement: B5.2 "I feel more productive using pair programming than programming on my own". These results were considered in relation to the qualitative results to see if the situation was clarified. The qualitative results depict some positive results with some concerns highlighted and thus also indicate some ambivalence in terms of how the developers feel about pair versus individual programming. The software developers indicate that when programming individually you have to do the whole job yourself, whereas with pair programming you get a team's effort, including the ideas and thoughts of another individual. The qualitative data shows that junior staff prefer to research and gain a better understanding of the issue on hand instead of being perceived as making a premature decision by a senior developer, which agrees with Tsyganok (2016).

The significant disagreement in the quantitative results relating to "I feel more productive using pair programming than programming on my own" contradicts previous studies that indicate software developers feel more productive when using pair programming due to the increases in knowledge sharing and improvement of quality (Begel & Nagappan, 2008; Kim et al., 2010; Williams, 2010). Although it was indicated in the quantitative results that the majority of the software developers are using pair programming in their current projects, they seem to indicate they are not at their most productive using this approach. The question of "productivity" was not explicitly explored in this study as it was not part of the sub themes in the conceptual framework constructs i.e. it was not made clear exactly what is meant by "productive". Productivity could relate to the speed at which code is developed, the quality of the code, improved teamwork, increased training etc. As this was not specified; detailing this factor will be considered for future research. This will require the questionnaire to be adjusted to include the details around perceived developer productivity in relation to using pair programming so that the construct 'productivity' can be more explicitly researched.

7.4 Subjective Norms

The research question that is aligned to subjective norms is "*How do subjective norms correlate to software developers" use of pair programming within an agile software development methodology?*" The sub themes of this construct are: senior staff and peer influence.

Senior Staff Influence

The sections C1.1 – C1.3 in the quantitative results are significant and respondents agree that senior staff influence is positive. The overall results of the qualitative data were also positive. The senior staff support the use of pair programming and encourage its use due to the benefits realised, namely: improving the quality of work and improved knowledge sharing.

The software developers are mostly paired as a junior and senior, as pair programming is mostly used as a training tool to help upskill developer and increase confidence, which agrees with Tsyganok (2016). Senior staff indicated that they notice a sense of reluctance from less experienced staff to participate e.g. not asking questions or adding input. A junior developer also indicated this, saying the reason is due to their lack of experience and knowledge. However other factors such as dominance and ego are experienced by junior developers when paired with a senior developer. Therefore, the correct pair mix needs to be considered in order for the software developers to feel comfortable and confident when pairing with a senior developer (Williams, 2006).

The senior staff adopt a hands-on approach: They would not spoon feed, but rather coach and guide the software developers in their learning. In addition the quantitative results depict that the senior staff provide formal opportunities to learn pair programming which enables a supportive culture. The positive influence of senior staff enables the practice of pair programming as the less experienced staff feel supported and encouraged. (Asaria et al., 2014).

Implications for practitioners: Senior staff influences

Since the senior staff support the use of pair programming, developers should focus on improving their interpersonal skills especially when there is a work level difference such as a junior and senior working together. The qualitative and quantitative data commonly flagged personality conflicts as an issue when using pair programming. Considering interventions to enhance peoples understanding of different personality types may be advantageous to software developers in a pair programming context. A possibility is to adopt personality workshops to assist in recognising the different personality types of staff in the company and how to proactively work together.

Peer Influence

The sections C2.1 – C2.3 in the quantitative data show respondents are significantly in agreement and the overall qualitative results were positive i.e. the peer influence on use of pair programming is positive. The software developers' stress in the quantitative and qualitative results that pair programming should be used as it reduces the amount of time spent on a task and improves the quality of software. This aligns with the literature that suggests that due to pair programming involving two individuals that can focus on writing code and another reviewing the code, they are able to deliver quality code in much less time than individual coding (Williams et al., 2000). In addition the software developers strongly feel that their peers like to use pair programming when it's applicable.

7.5 Perceived Behavioural Control

The research question that is aligned to perceived behavioural control is "*How does perceived behavioural control correlate to software developers" use of pair programming within an agile software development methodology?*" The sub themes of this construct are: resources/environmental factors and skill/abilities.

Resource/Environmental

The sections D1.1 – D1.2 in the quantitative results were significantly in agreement and qualitative results were also positive. The software developers were very positive about the existing office layout and hardware available for using pair programming. Therefore the company supports not only the right environment but sufficient resources to use pair programming. The software developers indicate that the company contains the right resources to use pair programming for instance they have an open plan office with wide, curved shaped desks that can fit two people and the correct hardware to support pair programming (Participant 1,4,5,6,7 and 8). They did note that a possible improvement could be to have more isolated rooms to reduce the noise level when using pair programming since there are many conversations and sometimes people talk loudly which may disturb others (Participant 1 and 6). In addition the management of the company is responsive and if the senior note that extra resources are required they are generally obtained, although they do sometimes require a motivation.

Implications for practitioners: Resource/Environmental factors

It is always important to have an environment that supports the use of a practice, especially if a practice vastly differs from the normal way of working. In this case open workspaces, large desks and appropriate hardware are necessary. However, since the developers have noted that the frequent interaction in pair programming could cause a disturbance to others, then noise suppression strategies such as desk divides between pairs to allow isolation and focused coding time for each day for a few hours may be desirable.

Skills/Abilities

The sections D2.1 – D2.2, D2.4 – D2.5 in the quantitative results show the software developers are positive about having the skills to write good code, test code and successfully debug code. For question D2.3 "I have been formally taught/trained to use pair programming" the results were not significant and are inconclusive. However, the developers appear to feel the skills/abilities they already have enabled them to use pair programming. The majority of the software developers have been using pair programming for more than one year. The company adopts their own internal process to improve the skills/abilities of their employees which is known as delivery practice. This is useful as it provides developers with opportunities to improve their skills, keeps the employees motivated and well aware of the latest technology.

In the qualitative results, the software developers felt that since pair programming involves lengthy interaction it becomes tiring and difficult to concentrate for too long therefore it becomes difficult to communicate their thoughts. The junior developers also noted that as part of communicating productively they would need to improve their listening skills which can help them gain constructive feedback and help in receiving constructive criticisms. Since the software developers are willing and eager to improve their skills/abilities it creates an energised, cross-skilled workforce.

Implications for practitioners: Skills/Abilities

Listening and communication skills need to be improved when using pair programming and increasing use of delivery practice. The company should adopt workshops that address potential flaws in current communication/listening skills and focuses on ways to improve these skills.

7.6 Behavioural Intention

The research question that is aligned to behavioural intention is "How does intention correlate to software developers" use of pair programming within an agile software development methodology?" The sub themes of this construct are: intent which is the intent to use pair programming.

The sections E1.1 – E1.3 in the quantitative results were significantly in agreement. The software developers felt very positive towards the use and future use of pair programming and, where possible, would increase its use. The qualitative results were somewhat positive but the software developers did have some concerns. Despite the benefits realised from pair programming. However the software developers have also noted personality differences can sometimes create a barrier to using pair programming. The software developers also noted that pair programming can become a burden as it requires a lot of energy and focus since it requires constant communication. Therefore software developers suggest taking frequent breaks.

Participants suggest pair programming reduces the time spent on a task. (Participant 7) and survey results show significant support for the idea that pair programming decreases time spent: B4.9 “feel pair programming produces high quality code faster” and C2.1 “My peers think pair programming should be used because it reduces the amount of time spent on a task”. While an individual participant felt it requires less effort pair programming is also perceived as more demanding because it requires more concentration and energy and for some individuals requires them to work outside their comfort zone. Therefore in question B5.2 “I feel more productive using pair programming than programming on my own” the software developers significantly disagreed that they felt productive when using pair programming as compared to individual programming. This could possibly be because it is a more demanding, difficult practice.

Pair programming is not suitable for all types of work. When senior developers use pair programming for simple tasks they find themselves feeling bored whereas for moderate to complex tasks, they find using pair programming beneficial. Participant 9, who is a senior, mentioned that with junior and senior staff pairing, it’s ideal to find the ‘flow’. The flow consists of where the less experienced are given tasks that are moderately complex but not highly complex. Since with a more complex task, less experienced developers find themselves ‘stuck in rabbit holes’. However, if senior staff are given a less complex task they find themselves ‘bored’.

Implications for practitioners: Intent

Pair programming becomes draining when conducted for lengthy periods of times (Kim et al., 2010). Therefore regular breaks must be taken and possibly swopping roles often to keep the momentum going during pair programming. In addition, the company needs to discover an ideal length of time to assign for pair programming that suits their software developers and then to encourage this scheduling as a way to avoid burnout or fatigue.

7.7 Conclusion

The attitude towards change and learning from both quantitative and qualitative results was significant and positive. The software developers are responsive towards change and can cope with changes. However changes that affect their way of working directly are sometimes more difficult to deal with. The software developers are very keen to learn. The company adopts an internal practice called delivery practice which assists to improve learning for all developers by using internal skills.

The results were significant and supported collaboration and participation. However the qualitative data highlighted challenges that were experienced such as interpersonal conflicts arising from having to adjust to conflicting personalities.

The quantitative results concerning attitude towards pair programming are inconclusive for question B4.5 (I feel pair programming reduces documentation), whereas the results from other related questions were positive. The qualitative results are positive stressing the development of higher quality code; increased knowledge sharing, which improves productivity, reduced time spent on delivering a task and an improvement of self-confidence. Pair programming has largely been used as a training tool to upskill less experienced staff by pairing senior and junior developers; however this did pose some challenges. The junior staff was reluctant to add input as they felt intimidated due to their lack of experience and that they were not formally taught/trained to use pair programming.

The quantitative results dealing with developers' preference of developing with pair programming versus individual programming were inconclusive. The qualitative results were positive with some concerns raised. The software developers indicate that when programming alone you have to struggle on the task alone whereas with pair programming you tend to have a team's effort. However the software developers indicate in the quantitative results that they disagree about feeling more productive using pair programming than programming on their own. This feeling could arise due to the issues which arise when using pair programming; such as personality conflicts, pair-related disturbances or lack of experience.

There were positive and significant results regarding senior and peer staff influence from both instruments. The senior staff displays a supportive culture towards the junior staff and emphasise guiding rather than spoon feeding the software developers. Peers are supportive and seem to enjoy the benefits of pair programming.

The results indicate significant agreement in relation to availability of resources and current environmental factors. The company undertakes an internal practice to minimise the disturbance by adopting a practice called focus time which emphasises more focus and less interruption. The software developers also mentioned that a divider between desks could reduce disturbance. The quantitative results are inconclusive for questions D2.3 (I have been formally taught/trained to use pair programming). Since the use of pair programming is mainly used as a training tool to upskill less experienced staff, they adopt an internal process called delivery practice that has a dedicated time for all software developers to learn and share knowledge at any work level. The software developers are positive towards using pair programming for current and future use. Due to the benefits realised from its use, it is deemed to be favourable to increase its current use.

The insight of relevance to practitioners was also highlighted in this chapter. However this was not the main focus of the study.

Chapter 8: Conclusion and Recommendation

8.1 Introduction

This chapter will discuss the concluding ideas from the qualitative and quantitative results per research question. In addition, it will discuss the limitations experienced in this study and avenues for further research.

8.2 Concluding remarks per Research Question

The adoption of agile in software development industries is widely increasing, however an alarming concern is that pair programming is one of the least adopted agile practice. The benefits of pair programming suggest that it aids a company to improve quality, productivity and team work. Pair programming is also seen to improve software developers on an individual level through the sharing of knowledge, through the continuous interaction and active participation involved when using pair programming. As the use of pair programming is limited, this study aims to understand what influences the adoption of pair programming in software development industries using agile methodology. The results for the research questions were obtained from quantitative and qualitative data in this study.

Research question 1: How do attitudes correlate to software developers' use of pair programming within an agile software development methodology?

The conceptual framework's construct 'attitude' is made up of a number of key components that fundamentally influence behaviour i.e. the use of pair programming. The argument for including the various aspects of attitude are based on the following rationale: The attitude towards change is influenced by an agile orientated industry which is often susceptible to changes and thus suggests the ability to cope with change is vital. To accommodate change software developers are continually learning in order to improve their abilities. When using pair programming there is frequent collaboration and participation which enables teamwork. By having a positive attitude this promotes team consensus and improves self-morale that inherently contributes to improved productivity. In addition the attitude towards pair programming influences how the developers engage with the practice: having a positive attitude towards pair programming motivates the software developers to overcome hurdles and engage with the process. However since pair programming practices are different from traditional programming, their preference for pair programming individually versus pair programming, influences their motivation when engaging with pair programming. The influence of attitude in this study is discussed below.

The positive aspects indicated by the software developers are that using pair programming within agile industries, learning is constant therefore the software developers are attuned to having a positive

attitude towards continued learning and adapting to changes. However some changes are difficult to deal with especially if daily duties are disrupted. There is an internal practice adopted called delivery focus to empower learning on pair programming practices and to keep them updated on the trends in the IT world.

In addition using pair programming, there is frequent collaboration and participation which allows for improved sharing of knowledge; high quality of work due to code errors or bugs being resolved earlier; and a decrease in the development time of work. The attitude towards pair programming is positive due to the benefits realised from the use, such as: positively influencing staff relations which increases self confidence and self-esteem that results in improved productivity and the improved quality of work and upskilling of staff. Pair programming in this company is commonly used as a mentor or training tool and helps less experienced staff boost their self-confidence. The attitude towards pair programming is positive as, in pair programming, there are always two minds involved on one task. The benefits of having a navigator role in pair programming is that it is often used to improve the quality of code since code errors and feedback are noted and resolved earlier.

However less experienced staff strongly noted feeling reluctant to voice themselves due to their lack of experience. In addition, having opposing personality types being involved in a pair causes conflicts which lead to debates and frustrations. It was indicated that having a junior paired with an extrovert could be advantageous in providing an overflow of information but can also be overwhelming. There are issues with pair related disturbances such as others speaking loudly, conversations being introduced that are not related to the task on hand etc., which breaks the flow of thoughts when using pair programming. The company adopts an internal practice called focus time to enforce minimal interruptions for a specified period of time.

The software developers indicate that they feel more productive using individual programming than pair programming. However since productivity was not a factor explored in this study, it is not clear exactly what they mean by "productivity" i.e. how the software developers individually would define and measure personal productivity. This does appear to contradict the above mentioned benefits they feel are realised when using pair programming but could be due to the negative factors mentioned which impact their work when using pair programming e.g. disruptions.

Research question 2: How do subjective norms correlate to software developers' use of pair programming within an agile software development methodology?

The senior staff displays a supportive culture that is willing to assist less experienced developers. This is also designed to empower them to learn and discover more by themselves by guiding them instead of spoon feeding. However as discussed in detail in section 7.4; the junior developers are not always

comfortable with sharing their thoughts, some of whom will prefer to look for more information on solving the problem on their own before engaging with a senior developer. However the peer relationships are supportive irrespective of work level and are always willing to share knowledge. The colleagues find the benefits of pair programming appealing, and if appropriate they would like to use pair programming. This creates a sense of conformity with other software developers in the company.

Research question 3: How does perceived behavioural control correlate to software developers' use of pair programming within an agile software development methodology?

The company makes use of large curved desks that can accommodate two individuals working on one desktop and have more than one keyboard or mouse plugged in so that when swopping roles there is no need to swop seats but just to take control of the appropriate hardware. In addition for the use of distributed pair programming the company also provides supporting tools and software such as Skype to ensure pair programming activities can still be performed. The office is set up as an open plan office so that agile methodologies and collaboration can easily be performed. However, due to pair programming involving a lot of conversations, the environment could be improved to avoid the noise levels that may disturb others.

The software developers find they are well skilled to code and review code; however the results of being formally trained to use pair programming were inconclusive. The software developers did note that the skills they would like to improve on, to support using pair programming, is to communicate more effectively such that dealing with conflict or criticism is improved and to develop good listening skills to attentively comprehend other's thoughts. This seems to suggest that developers require more soft (interpersonal) skills rather than hard (technical) skills

Research question 4: How does intention correlate to software developers' use of pair programming within an agile software development methodology?

The software developers find positive benefits of using pair programming encourage them to adopt its use more often in current and future projects. However there are still some factors that make the software developers continue to program individually. Some of the reasons as noted above that could make software developers hesitant to use pair programming, is the personality conflicts experienced and the pair-related disturbances when using pair programming that hinders its use. In addition the software developers indicate that pair programming is not always beneficial, for instance when a senior developer is using pair programming on simple tasks, they find themselves feeling bored, whereas with moderate to complex task pair programming is beneficial. Therefore with the different levels of skills and complexity of tasks, the benefit of pair programming will vary.

8.3 Limitations

Since the researcher used convenience sampling (detailed in chapter 4) there were few software development houses that met the selection criteria. This study contained a small sample size of only 17 individuals in one company. The reliability Cronbach Alpha result for the construct 'attitude towards collaboration and participation' was less than 0.7 (Alpha = 0.568), thus lower than the reliability threshold. This result could be low due to the small size of the population.

In this study there was less representation of females than males which represents a potential bias in the data. There were only 2 females from a total of 17 individuals that participated in the survey. The respondents for the interview were randomly selected by a Human Resource representative of the company, which did not include any female respondents. Therefore the researcher is unable to judge if the limited number of female perspectives has created any bias in the data.

The majority of the software developers (76.5%) have used pair programming for more than one year, therefore the responses in this study are based on a company that actively uses agile but also represents the views of software developers predominately experienced in using pair programming. There is thus a lack of the views from the perspective of developers who have only recently experienced attempting pair programming.

In addition, the software developers have noted mostly positive feedback which could potentially result in bias in the data i.e. item social desirability could be present. This means that the respondents' answers could be influenced by wanting to provide the socially desirable response such that they behave in an appropriate manner or conform to behaviour that they believe is generally accepted around them (Podsakoff et al., 2003). The potential for this bias is supported by the fact that senior staff is obviously supportive of pair programming and actively encourages its use.

8.4 Future Study

The study reported mainly on experienced pair programming software developers in this company that have been using this practice for more than one year. Therefore the views obtained represent the opinions of developers predominantly experienced in using pair programming. Future studies could also specifically include developers who have only been involved in pair programming for less than 6 months as they may be able to speak of the challenges being faced from their more recent experiences.

It was commonly flagged by the software developers that personality conflicts appear as an issue within both the quantitative and qualitative data when using pair programming. This includes factors relating to the various types of personalities like introverts, extroverts and dominants. The conceptual framework did not include personality as a construct of the model but this has often been commented on by respondents. This can therefore lead to a future study on the influences and impact personality

types have on using pair programming, by including it within the conceptual model based on the theory of planned behaviour.

The software developers also indicated within the quantitative results that they feel more productive programming alone than pair programming. This study did not explicitly consider the aspects that could be considered to make up 'productivity' from the perspective of software developers when using pair programming. A future study could focus on exploring this construct in more detail.

In addition, it was raised that when senior developers are using pair programing for simple tasks they find themselves bored, however if it involves a junior developer it may be beneficial as the junior developer will learn how to solve the task on hand. Therefore a possible future study could consider the need to, and benefit of, using pair programming dependent on some function of (a) the skill level of the developer and (b) complexity of task. This could include (c) considering the major reason the company has in wanting to employ pair programming e.g. improving performance, training and mentorship, creating better team skills or perhaps a combination of these benefits.

There were three inconclusive results indicated in the quantitative results as mentioned below. These factors could be investigated in other industry settings e.g. including both pair programming settings and those that do not use pair programming as the results may provide more detail by accessing a larger sample and provide significant results:

- B4.5 "I feel pair programming reduces documentation"
- B5.1 "I prefer using pair programming technique over individual programming technique"
- D2.3 "I have been formally taught/trained to use pair programming"

8.5 Conclusion

The company selected for this study is a software development house that proactively adopts agile practices and pair programming. Pair programming is seen as the normal culture of the company. The company makes use of an agile methodology to promote an adaptive environment to cope with the evolving needs. Pair programming is realised for its admirable benefits such as increased knowledge sharing, improving the quality of code, reducing the time spent on a task and improving the self-confidence of less experienced software developers. These benefits are embedded in the common usage of pair programming for the purpose of mentoring or upskilling less experienced staff and improving the quality of code. The company provides adequate environment and resource factors to use pair programming and the agile methodology helps emphasise the agile principles to empower the dynamics of pair programming such as increased interaction, collaboration and responding to changes.

Even in the presence of the benefits, there are some issues such as the constant interaction in pair programming which sometimes becomes disturbing for others. In addition, some personality types seem to be overpowering and overwhelming and that leads to conflict and debates. However the company supports the use of pair programming by adopting practical strategies to not only overcome the known challenges in pair programming but to also improve its use: Using internal practices to resolve these challenges, for example delivery practice to improve learning and skill/abilities development and focus time; used to minimise noise interruptions for a dedicated period of time.

The use of pair programming is more likely to be adopted by having the support and structures in place to support the dynamics. Agile helps to support the dynamics of pair programming via the principle of interacting and collaborating. The more agile the environment the more easily it is to use pair programming however the some developers find pair programming more useful in complex tasks than simple task. In addition there challenges experiences when dealing with complex personalities and the constant conversing in pair programming sometimes leads to conflict, therefore by advancing on interpersonal skills will assist with using pair programming more efficient and effectively. Lastly with a positive attitude, motivating subjective norms and ease of perceived behavioural control factors, the more the software developers will make use of pair programming.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Armitage, CJ. (2001). Efficacy of the Theory of Planned Behaviour: A meta-analytic review. *British Journal of Social Psychology*, 40(4), 471–499.

Asaria, M., Ruhollah, S., & Manucheh, R. (2014). *A Theoretical Model of Workforce Agility Based on the Theory of Planned Behavior*. Paper presented at the The 3th International Conference on Behavioral Science, Kish – IRAN. https://www.academia.edu/6253784/A_Theoretical_Model_of_Workforce_Agility_Based_on_the_Theory_of_Planned_Behavior plus access date (Accessed: 28 June 2018).

Ashcraft, C., McLain, B., & Eger, E. (2016). Women in tech: The facts. *National Center for Women & Information Technology (NCWIT)*.

van Aalst, J. (2010). Using Google Scholar to estimate the impact of journal articles in education. *Educational Researcher, 39*(5), 387-400. https://hub.hku.hk/bitstream/10722/129415/3/content.pdf?accept=1 (Accessed: 23 June 2018).

Balbes, M. (2014). Conflict and Resolution in the Agile World *The Agile Architect*. https://adtmag.com/articles/2014/12/17/agile-conflict-resolution.aspx (Accessed: 10 March 2013).

Balijepally, V., Mahapatra, R., Nerur, S., & Price, KH. (2009). Are two heads better than one for software development? The productivity paradox of pair programming. *MIS quarterly*, 91-118.

Begel, A., & Nagappan, N. (2008). *Pair programming: what's in it for me?* Paper presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, 120-128. http://andrewbegel.com/papers/esem-begel-2008.pdf (Accessed: 03 Feburary 2012).

Boehm, B. (2006). *A view of 20th and 21st century software engineering.* Paper presented at the Proceedings of the 28th international conference on Software engineering, 12-29. https://www.ida.liu.se/~729A40/exam/Barry%20Boehm%20A%20View%20of%2020th%20and%2021st%20Century%20Software%20Engineering.pdf (Accessed: 17 May 2012).

Cockburn, A., & Williams, L. (2000). The costs and benefits of pair programming. *Extreme programming examined*, 8, 223-247. http://www.cs.pomona.edu/~markk/cs121.f07/supp/williams_prpgm.pdf (Accessed: 19 June 2013).

Cohen, L., Manion, L., & Morrison, K. (2007). Research Methods in Education: The Higher Education Academy Innovation Way. *York Science Park: Heslington*.

Creswell, JW., & Miller, DL. (2000). Determining validity in qualitative inquiry. *Theory into practice*, *39*(3), 124-130.

Dhoodhanath, P. (2014). *Pair Programming and collaboration in software engineering course.* . (BComm Honors ISTN), University of KwaZulu-Natal (South Africa).

Dillenbourg, P. (1999). What do you mean by collaborative learning? *Collaborative-learning: Cognitive and Computational Approaches.*(1), 1-19.

Doyle, M., Williams, L., & Cohn, R. (2014). Agile software development in practice. Presented in international conference on Agile Processes in Software Engineering and Extreme Programming Agile software development in practice, 32-45. Springer, Cham. https://s3.amazonaws.com/academia.edu.documents/36577380/ESEMUpdate.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1530038187&Signature=tU4L2VmxFQkhhksT9Zn5%2FTtf0%2Bo%3D&response-content-disposition=inline%3B%20filename%3DAgile_Software_Development_in_Practice.pdf (Accessed: 19 April 2011).

Dudziak, T. (1999). Extreme programming an overview. *Methoden und Werkzeuge der Softwareproduktion WS, 2000*, 1-28. http://csis.pace.edu/~marchese/CS616/Agile/XP/XP_Overview.pdf (Accessed: 23 March 2012).

Fu, Q., Grady, F., Broberg, BF., Roberts, A., Martens, GG., Johansen, KV., & Loher, PL. (2017). Code review and cooperative pair programming best practice. *arXiv preprint arXiv:1706.02062*.

Icek, A. (1991). The theory of planned behavior. *Organizational behavior and human decision processes*, *50*(2), 179-211. https://s3.amazonaws.com/academia.edu.documents/32876859/Ajzen_1991_The_theory_of_planned_behavior.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1530038430&Signature=A0D1S%2FdP8Os%2FLf6MeAufp%2B%2BtPeo%3D&response-content-

disposition=inline%3B%20filename%3DAjzen_1991_The_theory_of_planned_behavio.pdf (Accessed: 25 April 2011).

Icek, A. (2006). The Theory of Planned Behavior. *Behavioral Change Models*. http://sphweb.bumc.bu.edu/otlt/MPH-Modules/SB/SB721-Models/SB721-Models3.html (Accessed: 15 March 2013).

Icek, A. (2015). Theory of planned behaviour. *TPB Diagram*. http://people.umass.edu/aizen/tpb.html (Accessed: 28 March 2012).

Kaminsky, D. (2008). Agile method and Extreme Programming: Differences and Similarities. *IT Passion IT Professional Blog*. https://mauriziostorani.wordpress.com/2008/07/04/agile-method-and-extreme-programming-differences-and-similarities/ (Accessed: 25 October 2012).

Kim, L., Barnes, M., Chan, KA., & Keith, CC. (2010). Pair programming: Issues and challenges. Paper in *Agile Software Development*, 143-163.Springer, Berline, Heidelberg.

Klawe, M., Whitney, T., & Simard, C. (2009). Women in computing-take 2. *Communications of the ACM, 52*(2), 68-76. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.683.4282&rep=rep1&type=pdf (Accessed: 23 March 2012).

Konovalov, S., & Misslinger, S. (2006). Extreme Programming. http://wwwmayr.in.tum.de/konferenzen/Jass06/courses/3/presentations/3.%20Extreme%20Programming/ExtremeProgramming.pdf (Accessed: 03 November 2014).

Kress, G. (2009). *Multimodality: A social semiotic approach to contemporary communication*. New York: Routledge.

Layman, L., Williams, L., Osborne, J., Berenson, S., Slaten, K., & Vouk, M. (2005). *How and Why Collaborative Software Development Impacts the Software Engineering Course.* Paper presented at the In Frontiers in Education, Proceedings 35th Annual Conference, T4C-T4C). IEEE. https://www.researchgate.net/profile/Kelli_Slaten/publication/4231977_How_and_Why_Collaborative_Software_Development_Impacts_the_Software_Engineering_Course/links/004635198ef0f8e292000000.pdf (Accessed: 19 March 2012).

Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. Information systems management, 21(3), 41-52.

Pallant, J. (2001). *SPSS Survival manual: a step by step guide to data analysis using SPSS for windows (versions 10 and 11): SPSS student version 11.0 for windows*: Open University Press.

Podsakoff, PM., MacKenzie, SB., Lee, J., & Podsakoff, NP. (2003). Common method biases in behavioral research: A critical review of the literature and recommended remedies test. *Journal of applied psychology*, *88*(5), 879.

Schmidt, C., Kude, T., Heinzl, A., & Mithas, S. (2014). How Agile practices influence the performance of software development teams: The role of shared mental models and backup. Paper presented at the Thirty Fifth International Conference on Information Systems, Auckland.

Sekaran, U., & Bougie, R. (2013). *Research methods for business: A skill building approach* (6 ed.). New York: John Wiley & Sons.

Standish, Group. (2013). CHAOS MANIFESTO 2013 Think Big, Act Small. http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf (Accessed: 17 November 2012).

Stankovic, D., Nikolic, V., Djordjevic, M., & Cao, DB. (2013). A survey study of critical success factors in agile software projects in former Yugoslavia IT companies. Journal of Systems and Software, 86(6), 1663-1678.T, Dudziak. (1999). Extreme programming an overview. *Methoden und Werkzeuge der Softwareproduktion WS, 2000*, 1-28.

Tao, D. (2008). *Using theory of reasoned action (TRA) in understanding selection and use of information resources: An information resource selection and use model(Doctoral dissertation,*. University of Missouri-Columbia.

Tashakkori, A, & Teddlie, C. (2010). *Sage handbook of mixed methods in social & behavioral research*: Sage (2ed). United Stated of America: Sage Publications, Inc.

Tolfo, C, & Wazlawick, RS. (2008). The influence of organizational culture on the adoption of extreme programming. *Journal of systems and software*, *81*(11), 1955-1967.

Tsyganok, I. (2016). *Pair-Programming from a Beginner's Perspective.* Paper presented at the International Conference on Agile Software Development, 270-277. Springer, Cham. https://www.researchgate.net/publication/303182580_Pair-Programming_from_a_Beginner%27s_Perspective/fulltext/573874d008aea45ee83eaace/303182580_Pair-Programming_from_a_Beginner%27s_Perspective.pdf?origin=publication_detail (Accessed: 28 June 2018).

Vanhanen, J., & Lassenius, C. (2005). *Effects of pair programming at the development team level: An experiment.* Paper presented at the Proceedings of the 4th International Symposium on Empirical Software Engineering, Australia. http://lib.tkk.fi/Diss/2011/isbn9789526044132/article5.pdf (Accessed: 28 June 2018).

Williams, L. (2010). Pair Programming. *Encyclopedia of Software Engineering, 2*. https://collaboration.csc.ncsu.edu/laurie/Papers/ESE%20WilliamsPairProgramming_V2.pdf (Accessed: 19 March 2012).

Williams, L., Kessler, R., Cunningham, W, & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE software, 17*(4), 19-25.

Williams, L., Layman L., Osborne J., Katira N. (2006). *Examining the compatibility of student pair programmers*. Paper presented at the Paper presented at the In Agile Conference 2006, Indianapolis. http://www.lucas.ezzoterik.com/papers/LLO06.pdf (Accessed: 29 October 2016).

Withagen C., Bruel, R. (2007). Nothing is as practical as a good theory. *Analysis of theories and a tool for developing interventions to influence energy-related behaviour* 1-16. http://www.cres.gr/behave/pdf/paper_final_draft_CE1309.pdf (Accessed: 13 October 2017).

Yin, RK. (2017). *Case study research and applications: Design and methods*: Sage publications. https://pdfs.semanticscholar.org/89c8/30dc397c4d76c8548b8f5f99def607798feb.pdf (Accessed: 04 May 2014).

Zaza, S., Wright-De Agüero, LK., Briss, PA., Truman, BI., Hopkins, DP., Hennessy, MH.,Sosin, DM., Anderson, L., Carande-Kullis, VG., Teutsch, SM., Pappaionou, M. (2000). Data collection instrument and procedure for systematic reviews in the guide to community preventive services. *American journal of preventive medicine, 18*(1), 44-74.

1

[1] APA 6th convention used for referencing

Appendix A: Node Report from NVivo

Node Report

Parent Node Name	Name	Description
	Agile structure	
	Ways to make pair programming work better	
	Use of pair programming within the company	
	Subjective Norms	
	race	
	Power Tool	Describes the defination of pair programming being a power tool
	Perception of others about pair programming	
	Perceived Behavioural Control	
	Mob sessions	
	Focus time	
	distributed geographic pairing	
	Distractions when using pair programming	
	Conflict mode	this is when you engaging with the pair and the conversation leads to conflict
	Challenges with team work	
	Behavioural Intention	
	Attitudes	

Parent Node Name	Name	Description
Nodes\\Attitudes		
	Developer's preference of developing with pair programming versus individual programming	
	Attitude towards pair programming	
	Attitude towards learning	
	Attitude towards collaboration and participation	
	Attitude towards change	
Nodes\\Attitudes\Attitude towards learning		
	Delivery practise	
Nodes\\Attitudes\Developer's preference of developing with pair programming versus individual programming		
	Individuals perceptive of working in a team	
Nodes\\Behavioural Intention		
	Intent of using pair programming	

Parent Node Name	Name	Description
Nodes\\Distractions when using pair programming		
	Conversation	
	Surrounding enviornment	
Nodes\\Perceived Behavioural Control		
	Skills and abilities	
	Resource or Environmental factors	
Nodes\\Perceived Behavioural Control\Skills and abilities		
	Driver role	
	Navigator role	
Nodes\\Subjective Norms		
	Peer influence	
	Senior Staff influence	

Parent Node Name	Name	Description
Nodes\\Use of pair programming within the company		
	Ways of using pair programming	This describes the instances pair programming is used
	Benefit of pair programming	
	Personality mix	
	Quality of work delivered	
	When is pair programming used	

Appendix B: Questionnaire Form

Questionnaire

UNIVERSITY OF KWAZULU-NATAL ™

INYUVESI YAKWAZULU-NATALI

UKZN HUMANITIES AND SOCIAL SCIENCES RESEARCH ETHICS COMMITTEE (HSSREC)

APPLICATION FOR ETHICS APPROVAL

For research with human participants

Information Sheet and Consent to Participate in Research

Dear Participant,

My name is Prashika Dhoodhanath from the School of Management, IT and Governance, College of Law and Management Studies, University of KwaZulu-Natal (Westville campus).

You are being invited to participate in a study that involves research on pair programming used by agile software development companies. The aim and purpose of this research is to gain a better understanding of pair programming in a software development environment. I will conduct questionnaires, interviews and review company documents. The duration of your participation if you choose to participate and remain in the study will be: to answer a questionnaire (approximately 20 minutes) and potentially to participate in an interview (approximately 1 hour). These should be completed within a month or two from commencement (depending on the availability of staff).

This study has been ethically reviewed and approved by the UKZN Humanities and Social Sciences Research Ethics Committee (HSS/1664/017M).

In the event of any problems or concerns you may contact the researcher at 0728364790 or prashika.dhoodhanath@gmail.com ; my supervisor at 0839790833 or Quillingr@ukzn.ac.za or the UKZN Humanities & Social Sciences Research Ethics Committee:

Questionnaire

Research Office, Westville Campus

Govan Mbeki Building
Private Bag X 54001
Durban 4000 KwaZulu-Natal, SOUTH AFRICA

Tel: 27 31 2604557- Fax: 27 31 2604609

Email: HSSREC@ukzn.ac.za

Your participation in the study is voluntary and by participating, you are granting the researcher permission to use your responses. You may refuse to participate or withdraw from the study at any time with no negative consequence. There will be no monetary gain from participating in the study. Your anonymity will be maintained by the researcher and the School of Management, I.T. & Governance and your responses will not be used for any purposes outside of this study.

All data, both electronic and hard copy will be securely stored during the study and archived for 5 years. After this time, all data will be destroyed.

If you have any questions or concerns about participating in the study, please contact me or my research supervisor at the numbers listed above.

Sincerely,

Prashika Dhoodhanath

(211503263)

Questionnaire

CONSENT TO PARTICIPATE

I (Name) _____ have been informed about the study entitled: Factors influencing software developers' use of pair programming in an agile software development methodology environment by Miss P Dhoodhanath (MCOM: IS&T).

I understand the purpose and procedures of the study; Factors influencing software developers' use of pair programming in an agile software development methodology environment.

I have been given an opportunity to ask questions about the study and have had answers to my satisfaction.

I declare that my participation in this study is entirely voluntary and that I may withdraw at any time without affecting any of the benefits that I usually am entitled to.

If I have any further questions or queries related to the study I understand that I may contact the researcher at: email: prashika.dhoodhanath@gmail.com or contact number: 0728364790.

If I have any questions or concerns about my rights as a study participant, or if I am concerned about an aspect of the study or the researchers then I may contact:

HUMANITIES & SOCIAL SCIENCES RESEARCH ETHICS ADMINISTRATION
Research Office, Westville Campus
Govan Mbeki Building
Private Bag X 54001
Durban
4000
KwaZulu-Natal, SOUTH AFRICA
Tel: 27 31 2604557 - Fax: 27 31 2604609
Email: HSSREC@ukzn.ac.za

Additional consent, where applicable. I hereby provide consent to:

Audio-record my interview	Yes	No
Video-record my interview / focus group discussion	Yes	No
Use of my photographs for research purposes (photos will be taken of the workplace and may include the participants)	Yes	No

_____ _____

Signature of Participant **Date**

Please provide the following details that will be used to arrange interviews (If necessary):

_____ _____

Email Address **Contact Number**

Questionnaire

Dear Participant,

This questionnaire forms part of my master's research entitled: *"Factors influencing software developers' use of pair programming in an agile software development methodology environment"* for the degree of MCOM (IS&T) at the University of KwaZulu-Natal.

The aim of this study is to investigate factors that influence the use of pair programming in an agile software development company. The findings of the study will be of benefit by enhancing our understanding of the phenomenon of pair programming and potentially allow developers to realise the inherent benefits of adopting the programming practise.

You are kindly requested to complete this survey questionnaire, comprising four sections as honestly and frankly as possible and according to your personal views and experience. No foreseeable risks are associated with the completion of the questionnaire which is for research purposes only.

You are requested to provide your name so that your responses can be linked to interview responses should you be interviewed as part of this study. Your anonymity will be ensured in any reporting on the study. All information obtained from this questionnaire will be used for research purposes only and will remain confidential. Your participation in this survey is voluntary. You may withdraw from answering this survey without penalty at any stage. After the completion of the study, an electronic summary of the findings of the research will be made available to you on request. Please note that any participant under the age of 18 years old will be excluded from the dataset.

Permission to undertake this survey has been granted by the University of KwaZulu-Natal and the Ethics Committee. If you have any research-related enquiries, they can be addressed directly to me or my supervisor. My contact details are: 0728364790 e-mails: prashikadhoodhanath@gmail.com and my supervisor can be reached at 0839790833, email: rosemaryquilling@gmail.com, Discipline of Information Systems and Technology, School of Management, IT and Governance, College of Law and Management Studies, UKZN.

Please return the completed questionnaire to Prashika Dhoodhanath by 06 October 2017.

1. Gender:

Male	Female

2. Age:

<20	20-25	26-30	31-35	36-40	>40

3. Total years of work experience:

<2years	2 - <3 years	3- <5 years	5+ years

4. Work level: ***Tick ALL that apply***

4.1Intern/Graduate	4.2 Junior Developer	4.3 Intermediate Developer	4.4 Senior Developer	4.5 Manager	4.6 Other(Specify) _____

5. Highest Formal Qualifications:

5.1 None	5.2 Certificate	5.3 Diploma	5.4 Degree	5.5 Honours	5.6 Masters	5.7 Doctorate	5.8 Other (Specify) _____

Questions

SECTION A: USE OF PAIR PROGRAMMING

A.1 How long have you been using pair programming?

Never used it	Less than 3 months	3 - <6 months	6 - <12 months	1 year +

Indicate your agreement with the following statement

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
A.2 I currently use pair programming in my projects					

SECTION B: ATTITUDE

Indicate your agreement with the following statements

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1. ATTITUDE TOWARDS CHANGE					
B1.1 I welcome change and can cope when changes are applied to methodology at any stage of development, even late in development					
B1.2 I welcome change to the process, even when it's the first time I experience something					
B1.3 I am happy to change the way I do programming if it results in better output					

Questionnaire

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
2.ATTITUDE TOWARDS LEARNING					
B2.1 I am interested in learning new things about design issues in software development					
B2.2 I will gladly learn a new method of programming					
B2.3 I will gladly learn about new programming tools or techniques					
3. ATTITUDE TOWARDS COLLABORATION AND PARTICIPATION					
B3.1 I am happy to follow the process defined and agreed upon by the team					
B3.2 I trust my team members to make decisions on my behalf					
B3.3 I am motivated by my team members to participate in the development processes					
B3.4 I prefer working in a team to working alone					
B3.5 I work well with others on a task					
4. ATTITUDE TOWARDS PAIR PROGRAMMING					
B4.1 I feel productive when using pair programming					
B4.2 I feel pair programming empowers knowledge sharing					
B4.3 I feel pair programming improves my self-motivation					
B4.4 I believe developers find and implement better code solutions by using pair programming					
B4.5 I feel pair programming reduces documentation					
B4.6 I feel errors are found earlier when using pair programming					
B4.7 I feel pair programming reduces the need for review/inspection of code					
B4.8 I feel pair programming produces high quality code					
B4.9 I feel pair programming produces high quality code faster					

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
5. DEVELOPER'S PREFERENCE OF DEVELOPING WITH PAIR PROGRAMMING VERSUS INDIVIDUAL PROGRAMMING					
B5.1 I prefer using pair programming technique over individual programming technique					
B5.2 I feel more productive using pair programming than programming on my own					
B5.3 I feel pair programming is effective in reducing the software development effort compared to individual programming					
B5.4 I feel the software delivered when using the pair programming technique is of a higher quality than individual programming					

SECTION C: SUBJECTIVE NORM

Indicate your agreement with the following statements

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1. SENIOR STAFF INFLUENCE					
C1.1 My superiors encourage the use of pair programming					
C1.2 My superior provides formal opportunities for the developers to learn pair programming					
C1.3 My superior feels pair programming improves the delivery of a product					
2.PEER INFLUENCE					
C2.1 My peers think pair programming should be used because it reduces the amount of time spent on a task					
C2.2 My peers think pair programming should be used because it improves the quality of software					
C2.3 My peers like to use pair programming whenever it is applicable					

SECTION D: PERCEIVED BEHAVIOURAL CONTROL

Indicate your agreement with the following statements

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1. RESOURCE/ENVIRONMENTAL					
D1.1 Pair programming can be done using the existing hardware available					
D1.2 Pair programming is possible with the existing layout of the office space					
2. SKILLS AND ABILITIES					
D2.1 I am able to write code correctly					
D2.2 I am able to test the code I write					
D2.3 I have been formally taught/trained to use pair programming					
D2.4 I have enough work experience and knowledge on pair programming					
D2.5 I have the skills to debug faulty programming code					

SECTION E: BEHAVIOURAL INTENT

Indicate your agreement with the following statements

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1. INTENTION					
E1.1 Given the choice, I will use pair programming in my current project, if applicable					
E1.2 Given the choice, I will increase my use of pair programming, where applicable					
E1.3 Given the choice, I will use pair programming in the future, where applicable					

If you have any other input to provide on Factors influencing software developers' use of pair programming in an agile software development methodology, please use this section for your comments:

Thank you for participating in this study.

Appendix C: Interview Schedule for Senior Staff

Interview Schedule for Senior Staff

UNIVERSITY OF KWAZULU-NATAL ™

INYUVESI YAKWAZULU-NATALI

Dear Participant,

This interview forms part of my master's research entitled: *"Factors influencing software developers' use of pair programming in an agile software development methodology environment"* for the degree of MCOM (IS&T) at the University of KwaZulu-Natal.

The aim of this study is to investigate factors that influence the use of pair programming in an agile software development company. The findings of the study will be of benefit by enhancing our understanding of the phenomenon of pair programming and potentially allow developers to realise the inherent benefits of adopting the programming practise.

You are kindly requested to complete this survey questionnaire, comprising four sections as honestly and frankly as possible and according to your personal views and experience. No foreseeable risks are associated with the completion of the questionnaire which is for research purposes only.

You are requested to provide your name so that your responses can be linked to the questionnaire responses should you be involved in participating in the questionnaires. Your participation in this survey is voluntary. You may withdraw from answering this survey without penalty at any stage. After the completion of the study, an electronic summary of the findings of the research will be made available to you on request.

Permission to undertake this survey has been granted by the University of KwaZulu-Natal and the Ethics Committee. If you have any research-related enquiries, they can be addressed directly to me or my supervisor. My contact details are: 0728364790 e-mails: prashikadhoodhanath@gmail.com and my supervisor can be reached at 0839790833, email: rosemaryquilling@gmail.com, Discipline of Information Systems and Technology, School of Management, IT and Governance, College of Law and Management Studies, UKZN.

Full Name and Surname: _____

Job Title: _____

Interview Date: _____

Interview Start Time: _____

Interview End Time: _____

Location of interview: _____

Interviewer: _____

QUESTIONS ASKED:

A. Attitude:

1. Does Chillisoft exclusively focus on agile methods or do they still incorporate traditional waterfall approaches?
2. Do you believe agile has been improving team relationships? If yes, to what extent? Why do you think this is important? If not, does this impact team work or productivity?
3. Do you use pair programming? To what extent would you say it is used within the company?
4. How does your company accommodate to the needs of pair programing? Describe the layout, techniques etc.
5. Are there any negative impacts of using pair programming in your company? To what extent?
6. Is there anything you think can be done better when using pair programming that can also assist to reduce the negative factors experienced?
7. As you work in an agile environment, do you believe using pair programming benefits the organisation? In what way and to what extent
8. Does pair programming improve team delivery of a product? How does this work?
9. Do you see pair programming contributing to improving team collaboration and participation? In what way and to what extent? If not, does this impact team work or productivity?
10. When necessary, how do you encourage change within a team?
11. Do you find Chillisoft teams are responsive to change? Are their specific situations where teams are not responsive to change?
12. Does Chilisoft encourage developers to learn pair programming? If yes then how do you achieve this? If no, is there any reason you would not encourage developers to learn pair programming?
13. Does Chillisoft provide formal opportunities for staff learning? Are there any specific programmes or training for developers? Does Chillisoft pay for training?

B. Subjective Norm

1. Do you see pair programming as the norm or the exception within Chillisoft?
2. Do you see pair programming as the norm or the exception within other software development organisations?
3. Do you think the team solely relies on what you suggest as a manager? To what extent do you think you influence their decisions?
4. Do you think you are influenced by peers when making decisions that impact the team? To what extent do you use their input when making a decision?

C. Perceived behavioural control

1. To what degree do you as part f senior staff oversee the securing of resources/environmental factors at Chillisoft? If yes, then how do you do it? If not, would you speak on behalf of the team if you feel they require more resources?
2. How do you improve the skills/abilities of employees in order to enhance their ability to understand the need for a specific process?
3. Are you able to authorise the necessary purchases?

D. Behavioural intent

1. Do you feel pair programming should be promoted within the organisation for current and future use? Yes/ No?
   WHY? Do you have evidence of it being useful/ making no difference (or even having a negative impact)?

Appendix D: Interview Schedule for Developers

Interview Schedule for Developer

UNIVERSITY OF KWAZULU-NATAL ™

INYUVESI YAKWAZULU-NATALI

Dear Participant,

This interview forms part of my master's research entitled: *"Factors influencing software developers' use of pair programming in an agile software development methodology environment"* for the degree of MCOM (IS&T) at the University of KwaZulu-Natal.

The aim of this study is to investigate factors that influence the use of pair programming in an agile software development company. The findings of the study will be of benefit by enhancing our understanding of the phenomenon of pair programming and potentially allow developers to realise the inherent benefits of adopting the programming practise.

You are kindly requested to complete this survey questionnaire, comprising four sections as honestly and frankly as possible and according to your personal views and experience. No foreseeable risks are associated with the completion of the questionnaire which is for research purposes only.

You are requested to provide your name so that your responses can be linked to the questionnaire responses should you be involved in participating in the questionnaires. Your participation in this survey is voluntary. You may withdraw from answering this survey without penalty at any stage. After the completion of the study, an electronic summary of the findings of the research will be made available to you on request.

Permission to undertake this survey has been granted by the University of KwaZulu-Natal and the Ethics Committee. If you have any research-related enquiries, they can be addressed directly to me or my supervisor. My contact details are: 0728364790 e-mails: prashikadhoodhanath@gmail.com and my supervisor can be reached at 0839790833, email: rosemaryquilling@gmail.com, Discipline of Information Systems and Technology, School of Management, IT and Governance, College of Law and Management Studies, UKZN.

Full Name and Surname: _____

Job Title: _____

Interview Date: _____

Interview Start Time: _____

Interview End Time: _____

Location of interview: _____

Interviewer: _____

QUESTIONS ASKED:

A. Attitude:

1. Do you use pair programming? To what extent?
2. To what extent would you say pair programming is used within the company?
3. Do you feel pair programming has made a difference to the work you deliver? If so how and to what extent?
4. Are there any challenges with using pair programming that negatively impacts your performance?
5. What can be done differently when using pair programming to reduce the challenges you face?
6. Do you feel pair programming influences your relationship with team members? If so, to what extent? Is it positive or negative?
7. How do you feel about working in a team versus working alone when doing development?
8. Does working in a team as part of agile motivate you? If so, how does working in a team motivate you? If not, please explain
9. Is there anything about working in a team that hinders your performance?
10. How do you cope with change? What changes are easy to adjust to? Which are more difficult?
11. How do you react to change? Do you feel it plays an important role as a developer?
12. Does your work require you to learn new skills?

B. Subjective Norm

1. How do you feel you are influenced by the senior staff?
2. Do you challenge senior decisions if they don't appear to make logical sense? If so, to what extent are your views taken into account?
3. Is there any colleague at work that you see as a mentor or someone you can learn from? If so, to what extent do you rely on this person's views?

C. Perceived behavioural control

1. What skill or ability within your team would need to improve in order to use pair programming?
2. Can you use pair programming with the resources you currently have available?
3. How can your working environment be improved to use pair programming?

4. Behavioural intent

1. Do you feel pair programming should be promoted within the organisation for current and future use? How? And Why?

Appendix E: Alignment Matrix

Note: The actual question within the alignment matrix only refers to the question number. The actual questions of the survey can be found in Appendix B. The actual questions of the interviews can be found in Appendices C and D.

Main Research Question	Variables in Research Question	Measurement of Variable of Survey	Measurement of Variable of Senior Staff Interview	Measurement of Variable of Developer Interview	Survey Question no.	Senior Staff InterviewQuestion No.	Developer Interview Question No.
Profile					1		
					2		
					3		
					4		
					5		
1. Use of pair programming	Use	How **long** have you been **using pair programming**?	Pair programming ***experience***		A.1	A.3	A.1
					A.2	A.4	A.2
				Factors relating to working with a ***partner***			A.6

2. How do attitudes influence software developers' use of pair programming within an agile software development methodology?	Attitude	*Attitude towards* ***change***			B1.1	A.11	A.11
					B1.2	A.10	A.10
					B1.3		
		Attitude towards ***learning***			B2.1	A.12	A.12
					B2.2	A.13	
					B2.3		
		Attitudes towards ***collaboration*** *and* ***participation***			B3.1	A.1	A.9
					B3.2	A.2	A.8
					B3.3	A.8	
					B3.4	A.9	
					B3.5		
		Attitude towards ***pair programming***			B4.1	A.5	A.3
					B4.2	A.6	A.4
					B4.3	A.7	A.5
					B4.4		
					B4.5		
					B4.6		
					B4.7		
					B4.8		
					B4.9		
		Developer's preference of developing with ***pair programming vs individual programming***		Developer's preference of developing ***individually vs pair programming***	B5.1		A.7
					B5.2		
					B5.3		
					B5.4		

3. How do subjective norms influence software developers' use of pair programming within an agile software development methodology?	Subjective Norm	*Senior Staff* influence			C1.1	B.1	B.1
					C1.2	B.2	B.2
					C1.3	B.3	
		Peer influence			C2.1	B.4	B.3
					C2.2		
					C2.3		
4. How does perceived behavioural control influence software developers' use of pair programming within an agile software development methodology?	Perceived behavioural control	*ResourceEnvironmental*			D1.1	C.1	C.2
					D1.2	C.3	C.3
		Skills and Abilities			D2.1	C.2	C.1
					D2.2		
					D2.3		
					D2.4		
					D2.5		
6. How does intent (or intention?) influence software developers' use of pair programming within an agile software development methodology	Behavioural intent	*Intention*			E1.1	D.1	D.1
					E1.2		
					E1.3		

Appendix F: Survey Statistical Results

Note: The results depicted below are output from SPSS

Section A: Demographics Results:

1 Gender

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Male	15	88.2	88.2	88.2
	Female	2	11.8	11.8	100.0
	Total	17	100.0	100.0	

2 Age

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	21-25	5	29.4	29.4	29.4
	26-30	3	17.6	17.6	47.1
	31-35	5	29.4	29.4	76.5
	36-40	2	11.8	11.8	88.2
	>40	2	11.8	11.8	100.0
	Total	17	100.0	100.0	

3 Experience

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	<2 years	3	17.6	17.6	17.6
	2 - <3 years	3	17.6	17.6	35.3
	3 - <5 years	1	5.9	5.9	41.2
	5+ years	10	58.8	58.8	100.0
	Total	17	100.0	100.0	

4.1Intern/Graduate

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	No	17	100.0	100.0	100.0

4.2 Junior Developer

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	7	41.2	41.2	41.2
	No	10	58.8	58.8	100.0
	Total	17	100.0	100.0	

4.3 Intermediate Developer

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	3	17.6	17.6	17.6
	No	14	82.4	82.4	100.0
	Total	17	100.0	100.0	

4.4 Senior Developer

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	7	41.2	41.2	41.2
	No	10	58.8	58.8	100.0
	Total	17	100.0	100.0	

4.5 Manager

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yes	4	23.5	23.5	23.5
	No	13	76.5	76.5	100.0
	Total	17	100.0	100.0	

4.6 Other

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	No	17	100.0	100.0	100.0

5 Qualification

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Certificate	6	35.3	35.3	35.3
	Degree	8	47.1	47.1	82.4
	Honours	2	11.8	11.8	94.1
	Masters	1	5.9	5.9	100.0
	Total	17	100.0	100.0	

Section B: Use of Pair programming

Question: A.1 How long have you been using pair programming?

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	3 - <6 months	1	5.9	5.9	5.9
	6 - <12 months	3	17.6	17.6	23.5
	1+years	13	76.5	76.5	100.0
	Total	17	100.0	100.0	

Chi square goodness for fit result for question A.1 how long have you been using pair programming?

	Observed N	Expected N	Residual
3 - <6 months	1	5.7	-4.7
6 - <12 months	3	5.7	-2.7
1+years	13	5.7	7.3
Total	17		

Chi square Test Statistics

	A.1 How long have you been using pair programming?
Chi-Square	14.588[a]
df	2
Asymp. Sig.	.001

a. 0 cells (.0%) have expected frequencies less than 5. The minimum expected cell frequency is 5.7.

Question: A.2 "I currently use pair programming in my projects"

A.2 I currently use pair programming in my projects

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Strongly disagree	1	5.9	5.9	5.9
	Disagree	1	5.9	5.9	11.8
	Neutral	1	5.9	5.9	17.6
	Agree	10	58.8	58.8	76.5
	Strongly agree	4	23.5	23.5	100.0
	Total	17	100.0	100.0	

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
A.2 I currently use pair programming in my projects	17	3.88	1.054	.256

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
A.2 I currently use pair programming in my projects	3.453	16	.003	.882	.34	1.42

Section B: Attitude

Questions: B1.1 – B5.4

B1.1 I welcome change and can cope when changes are applied to methodology at any stage of development, even late in development

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Agree	10	58.8	58.8	64.7
	Strongly agree	6	35.3	35.3	100.0
	Total	17	100.0	100.0	

B1.2 I welcome change to the process, even when it's the first time I experience something

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	3	17.6	17.6	17.6
	Agree	10	58.8	58.8	76.5
	Strongly agree	4	23.5	23.5	100.0
	Total	17	100.0	100.0	

B1.3 I am happy to change the way I do programming if it results in better output

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	7	41.2	41.2	41.2
	Strongly agree	10	58.8	58.8	100.0
	Total	17	100.0	100.0	

B2.1 I am interested in learning new things about design issues in software development

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	6	35.3	35.3	35.3
	Strongly agree	11	64.7	64.7	100.0
	Total	17	100.0	100.0	

B2.2 I will gladly learn a new method of programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	1	5.9	5.9	5.9
	Agree	6	35.3	35.3	41.2
	Strongly agree	10	58.8	58.8	100.0
	Total	17	100.0	100.0	

B2.3 I will gladly learn about new programming tools or techniques

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	5	29.4	29.4	29.4
	Strongly agree	12	70.6	70.6	100.0
	Total	17	100.0	100.0	

B3.1 I am happy to follow the process defined and agreed upon by the team

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	1	5.9	5.9	5.9
	Agree	7	41.2	41.2	47.1
	Strongly agree	9	52.9	52.9	100.0
	Total	17	100.0	100.0	

B3.2 I trust my team members to make decisions on my behalf

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	3	17.6	17.6	17.6
	Agree	10	58.8	58.8	76.5
	Strongly agree	4	23.5	23.5	100.0
	Total	17	100.0	100.0	

B3.3 I am motivated by my team members to participate in the development processes

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	2	11.8	11.8	11.8
	Agree	8	47.1	47.1	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

B3.4 I prefer working in a team to working alone

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Neutral	6	35.3	35.3	41.2
	Agree	4	23.5	23.5	64.7
	Strongly agree	6	35.3	35.3	100.0
	Total	17	100.0	100.0	

B3.5 I work well with others on a task

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	3	17.6	17.6	17.6
	Agree	10	58.8	58.8	76.5
	Strongly agree	4	23.5	23.5	100.0
	Total	17	100.0	100.0	

B4.1 I feel productive when using pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	3	17.6	17.6	17.6
	Agree	5	29.4	29.4	47.1
	Strongly agree	9	52.9	52.9	100.0
	Total	17	100.0	100.0	

B4.2 I feel pair programming empowers knowledge sharing

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	5	29.4	29.4	29.4
	Strongly agree	12	70.6	70.6	100.0
	Total	17	100.0	100.0	

B4.3 I feel pair programming improves my self-motivation

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Neutral	2	11.8	11.8	17.6
	Agree	7	41.2	41.2	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

B4.4 I believe developers find and implement better code solutions by using pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	6	35.3	35.3	35.3
	Strongly agree	11	64.7	64.7	100.0
	Total	17	100.0	100.0	

B4.5 I feel pair programming reduces documentation

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	3	17.6	17.6	17.6
	Neutral	7	41.2	41.2	58.8
	Agree	4	23.5	23.5	82.4
	Strongly agree	3	17.6	17.6	100.0
	Total	17	100.0	100.0	

B4.6 I feel errors are found earlier when using pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Agree	4	23.5	23.5	29.4
	Strongly agree	12	70.6	70.6	100.0
	Total	17	100.0	100.0	

B4.7 I feel pair programming reduces the need for review/inspection of code

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Strongly disagree	1	5.9	5.9	5.9
	Disagree	2	11.8	11.8	17.6
	Neutral	2	11.8	11.8	29.4
	Agree	4	23.5	23.5	52.9
	Strongly agree	8	47.1	47.1	100.0
	Total	17	100.0	100.0	

B4.8 I feel pair programming produces high quality code

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	2	11.8	11.8	11.8
	Agree	7	41.2	41.2	52.9
	Strongly agree	8	47.1	47.1	100.0
	Total	17	100.0	100.0	

B4.9 I feel pair programming produces high quality code faster

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Strongly disagree	1	5.9	5.9	5.9
	Neutral	3	17.6	17.6	23.5
	Agree	6	35.3	35.3	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

B5.1 I prefer using pair programming technique over individual programming technique

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	2	11.8	11.8	11.8
	Neutral	8	47.1	47.1	58.8
	Agree	5	29.4	29.4	88.2
	Strongly agree	2	11.8	11.8	100.0
	Total	17	100.0	100.0	

B5.2 I feel more productive using pair programming than programming on my own

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	3	17.6	17.6	17.6
	Neutral	4	23.5	23.5	41.2
	Agree	9	52.9	52.9	94.1
	Strongly agree	1	5.9	5.9	100.0
	Total	17	100.0	100.0	

B5.3 I feel pair programming is effective in reducing the software development effort compared to individual programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Neutral	2	11.8	11.8	17.6
	Agree	11	64.7	64.7	82.4
	Strongly agree	3	17.6	17.6	100.0
	Total	17	100.0	100.0	

B5.4 I feel the software delivered when using pair programming technique is of a higher quality than individual programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Neutral	3	17.6	17.6	23.5
	Agree	8	47.1	47.1	70.6
	Strongly agree	5	29.4	29.4	100.0
	Total	17	100.0	100.0	

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
B1.1 I welcome change and can cope when changes are applied to methodology at any stage of development, even late in development	17	4.24	.752	.182
B1.2 I welcome change to the process, even when it's the first time I experience something	17	4.06	.659	.160
B1.3 I am happy to change the way I do programming if it results in better output	17	4.59	.507	.123
B2.1 I am interested in learning new things about design issues in software development	17	4.65	.493	.119
B2.2 I will gladly learn a new method of programming	17	4.53	.624	.151
B2.3 I will gladly learn about new programming tools or techniques	17	4.71	.470	.114
B3.1 I am happy to follow the process defined and agreed upon by the team	17	4.47	.624	.151
B3.2 I trust my team members to make decisions on my behalf	17	4.06	.659	.160
B3.3 I am motivated by my team members to participate in the development processes	17	4.29	.686	.166
B3.4 I prefer working in a team to working alone	17	3.88	.993	.241
B3.5 I work well with others on a task	17	4.06	.659	.160
B4.1 I feel productive when using pair programming	17	4.35	.786	.191
B4.2 I feel pair programming empowers knowledge sharing	17	4.71	.470	.114
B4.3 I feel pair programming improves my self-motivation	17	4.18	.883	.214
B4.4 I believe developers find and implement better code solutions by using pair programming	17	4.65	.493	.119

B4.5 I feel pair programming reduces documentation	17	3.41	1.004	.243
B4.6 I feel errors are found earlier when using pair programming	17	4.59	.795	.193
B4.7 I feel pair programming reduces the need for review/inspection of code	17	3.94	1.298	.315
B4.8 I feel pair programming produces high quality code	17	4.35	.702	.170
B4.9 I feel pair programming produces high quality code faster	17	4.06	1.088	.264
B5.1 I prefer using pair programming technique over individual programming technique	17	3.41	.870	.211
B5.2 I feel more productive using pair programming than programming on my own	17	3.47	.874	.212
B5.3 I feel pair programming is effective in reducing the software development effort compared to individual programming	17	3.94	.748	.181
B5.4 I feel the software delivered when using pair programming technique is of a higher quality than individual programming	17	4.00	.866	.210

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
B1.1 I welcome change and can cope when changes are applied to methodology at any stage of development, even late in development	6.769	16	.000	1.235	.85	1.62
B1.2 I welcome change to the process, even when it's the first time I experience something	6.628	16	.000	1.059	.72	1.40
B1.3 I am happy to change the way I do programming if it results in better output	12.908	16	.000	1.588	1.33	1.85
B2.1 I am interested in learning new things about design issues in software development	13.786	16	.000	1.647	1.39	1.90
B2.2 I will gladly learn a new method of programming	10.101	16	.000	1.529	1.21	1.85
B2.3 I will gladly learn about new programming tools or techniques	14.976	16	.000	1.706	1.46	1.95
B3.1 I am happy to follow the process defined and agreed upon by the team	9.713	16	.000	1.471	1.15	1.79
B3.2 I trust my team members to make decisions on my behalf	6.628	16	.000	1.059	.72	1.40
B3.3 I am motivated by my team members to participate in the development processes	7.778	16	.000	1.294	.94	1.65

B3.4 I prefer working in a team to working alone	3.665	16	.002	.882	.37	1.39
B3.5 I work well with others on a task	6.628	16	.000	1.059	.72	1.40
B4.1 I feel productive when using pair programming	7.098	16	.000	1.353	.95	1.76
B4.2 I feel pair programming empowers knowledge sharing	14.976	16	.000	1.706	1.46	1.95
B4.3 I feel pair programming improves my self-motivation	5.494	16	.000	1.176	.72	1.63
B4.4 I believe developers find and implement better code solutions by using pair programming	13.786	16	.000	1.647	1.39	1.90
B4.5 I feel pair programming reduces documentation	1.692	16	.110	.412	-.10	.93
B4.6 I feel errors are found earlier when using pair programming	8.235	16	.000	1.588	1.18	2.00
B4.7 I feel pair programming reduces the need for review/inspection of code	2.991	16	.009	.941	.27	1.61
B4.8 I feel pair programming produces high quality code	7.948	16	.000	1.353	.99	1.71
B4.9 I feel pair programming produces high quality code faster	4.012	16	.001	1.059	.50	1.62
B5.1 I prefer using pair programming technique over individual programming technique	1.951	16	.069	.412	-.04	.86
B5.2 I feel more productive using pair programming than programming on my own	2.219	16	.041	.471	.02	.92
B5.3 I feel pair programming is effective in reducing the software development effort compared to individual programming	5.191	16	.000	.941	.56	1.33
B5.4 I feel the software delivered when using pair programming technique is of a higher quality than individual programming	4.761	16	.000	1.000	.55	1.45

Section C: Subjective Norm

Questions C1.1 - C2.3

C1.1 My superiors encourage the use of pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	1	5.9	5.9	5.9
	Agree	9	52.9	52.9	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

C1.2 My superior provides formal opportunities for the developers to learn pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	10	58.8	58.8	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

C1.3 My superior feels pair programming improves the delivery of a product

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	2	11.8	11.8	11.8
	Agree	10	58.8	58.8	70.6
	Strongly agree	5	29.4	29.4	100.0
	Total	17	100.0	100.0	

C2.1 My peers think pair programming should be used because it reduces the amount of time spent on a task

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	7	41.2	41.2	41.2
	Agree	7	41.2	41.2	82.4
	Strongly agree	3	17.6	17.6	100.0
	Total	17	100.0	100.0	

C2.2 My peers think pair programming should be used because it improves the quality of software

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Neutral	2	11.8	11.8	17.6
	Agree	8	47.1	47.1	64.7
	Strongly agree	6	35.3	35.3	100.0
	Total	17	100.0	100.0	

C2.3 My peers like to use pair programming whenever it is applicable

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	1	5.9	5.9	5.9
	Agree	11	64.7	64.7	70.6
	Strongly agree	5	29.4	29.4	100.0
	Total	17	100.0	100.0	

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
C1.1 My superiors encourage the use of pair programming	17	4.35	.606	.147
C1.2 My superior provides formal opportunities for the developers to learn pair programming	17	4.41	.507	.123
C1.3 My superior feels pair programming improves the delivery of a product	17	4.18	.636	.154
C2.1 My peers think pair programming should be used because it reduces the amount of time spent on a task	17	3.76	.752	.182
C2.2 My peers think pair programming should be used because it improves the quality of software	17	4.12	.857	.208
C2.3 My peers like to use pair programming whenever it is applicable	17	4.24	.562	.136

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
C1.1 My superiors encourage the use of pair programming	9.200	16	.000	1.353	1.04	1.66
C1.2 My superior provides formal opportunities for the developers to learn pair programming	11.474	16	.000	1.412	1.15	1.67
C1.3 My superior feels pair programming improves the delivery of a product	7.628	16	.000	1.176	.85	1.50
C2.1 My peers think pair programming should be used because it reduces the amount of time spent on a task	4.190	16	.001	.765	.38	1.15
C2.2 My peers think pair programming should be used because it improves the quality of software	5.374	16	.000	1.118	.68	1.56
C2.3 My peers like to use pair programming whenever it is applicable	9.058	16	.000	1.235	.95	1.52

Section D: Perceived Behavioural Control

Questions D1.1 – D2.5

D1.1 Pair programming can be done using the existing hardware available

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	10	58.8	58.8	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

D1.2 Pair programming is possible with the existing layout of the office space

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	8	47.1	47.1	47.1
	Strongly agree	9	52.9	52.9	100.0
	Total	17	100.0	100.0	

D2.1 I am able to write code correctly

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	10	58.8	58.8	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

D2.2 I am able to test the code I write

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	9	52.9	52.9	52.9
	Strongly agree	8	47.1	47.1	100.0
	Total	17	100.0	100.0	

D2.3 I have been formally taught/trained to use pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Strongly disagree	1	5.9	5.9	5.9
	Disagree	3	17.6	17.6	23.5
	Neutral	2	11.8	11.8	35.3
	Agree	7	41.2	41.2	76.5
	Strongly agree	4	23.5	23.5	100.0
	Total	17	100.0	100.0	

D2.4 I have enough work experience and knowledge on pair programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Disagree	1	5.9	5.9	5.9
	Neutral	3	17.6	17.6	23.5
	Agree	8	47.1	47.1	70.6
	Strongly agree	5	29.4	29.4	100.0
	Total	17	100.0	100.0	

D2.5 I have the skills to debug faulty programming code

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	8	47.1	47.1	47.1
	Strongly agree	9	52.9	52.9	100.0
	Total	17	100.0	100.0	

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
D1.1 Pair programming can be done using the existing hardware available	17	4.41	.507	.123
D1.2 Pair programming is possible with the existing layout of the office space	17	4.53	.514	.125
D2.1 I am able to write code correctly	17	4.41	.507	.123
D2.2 I am able to test the code I write	17	4.47	.514	.125
D2.3 I have been formally taught/trained to use pair programming	17	3.59	1.228	.298
D2.4 I have enough work experience and knowledge on pair programming	17	4.00	.866	.210
D2.5 I have the skills to debug faulty programming code	17	4.53	.514	.125

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
D1.1 Pair programming can be done using the existing hardware available	11.474	16	.000	1.412	1.15	1.67
D1.2 Pair programming is possible with the existing layout of the office space	12.257	16	.000	1.529	1.26	1.79
D2.1 I am able to write code correctly	11.474	16	.000	1.412	1.15	1.67
D2.2 I am able to test the code I write	11.785	16	.000	1.471	1.21	1.74
D2.3 I have been formally taught/trained to use pair programming	1.975	16	.066	.588	-.04	1.22
D2.4 I have enough work experience and knowledge on pair programming	4.761	16	.000	1.000	.55	1.45
D2.5 I have the skills to debug faulty programming code	12.257	16	.000	1.529	1.26	1.79

Section E: Behavioural Intent

Questions E1.1 – E1.3

E1.1 Given the choice, I will use pair programming in my current project, if applicable

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	2	11.8	11.8	11.8
	Agree	8	47.1	47.1	58.8
	Strongly agree	7	41.2	41.2	100.0
	Total	17	100.0	100.0	

E1.2 Given the choice, I will increase my use of pair programming, where applicable

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Neutral	1	5.9	5.9	5.9
	Agree	10	58.8	58.8	64.7
	Strongly agree	6	35.3	35.3	100.0
	Total	17	100.0	100.0	

E1.3 Given the choice, I will using pair programming in the future, where applicable

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Agree	8	47.1	47.1	47.1
	Strongly agree	9	52.9	52.9	100.0
	Total	17	100.0	100.0	

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
E1.1 Given the choice, I will use pair programming in my current project, if applicable	17	4.29	.686	.166
E1.2 Given the choice, I will increase my use of pair programming, where applicable	17	4.29	.588	.143
E1.3 Given the choice, I will using pair programming in the future, where applicable	17	4.53	.514	.125

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
E1.1 Given the choice, I will use pair programming in my current project, if applicable	7.778	16	.000	1.294	.94	1.65
E1.2 Given the choice, I will increase my use of pair programming, where applicable	9.077	16	.000	1.294	.99	1.60
E1.3 Given the choice, I will using pair programming in the future, where applicable	12.257	16	.000	1.529	1.26	1.79

Appendix G: Reliability Statistical Results

Note: The results depicted below are output from SPSS

Cronbach's alpha results

Construct	Sub-scale	Items	Alpha
Attitude		B1.1 - 5.4	0.912
	Change	B1.1 - 1.3	0.701
	Learning	B2.1 - 2.3	0.894
	Collaboration & Participation	B3.1, 3.3- 3.5	0.568
	Pair Programming	B4.1 - 4.9	0.913
	Developers preference developing with pair programming versus individual programming	B5.1 - 5.4	0.796
Subjective Norm		C1.1 - 2.3	0.8
	Senior staff influence	C1.1 - 1.2	0.828
	Peer Influence	C2.1 - 2.3	0.804
Perceived Behavioural Control		D1.1 - 2.5	0.758
	Resource/Enviornmental	D1.1 - 1.2	0.882
	Skills	D2.1 - 2.2, 2.4-2.5	0.769
Behavioural Intention		E1.1 - 1.3	0.848

Appendix H: Correlation Statistical Results

Note: The results depicted below are output from SPSS

		BI	A.2 I currently use pair programming in my projects
ATT_CHANGE	Pearson Correlation	.651**	.145
	Sig. (2-tailed)	.005	.578
	N	17	17
ATT_LEARNING	Pearson Correlation	-.103	.031
	Sig. (2-tailed)	.695	.905
	N	17	17
ATT_CP	Pearson Correlation	.788**	.131
	Sig. (2-tailed)	.000	.615
	N	17	17
ATT_PP	Pearson Correlation	.641**	.103
	Sig. (2-tailed)	.006	.694
	N	17	17
ATT_PREFER	Pearson Correlation	.663**	.417
	Sig. (2-tailed)	.004	.096
	N	17	17

Spearman's rho	A.1 How long have you been using pair programming?	Correlation Coefficient	1.000
		Sig. (2-tailed)	.
		N	17
	SUBNORM_SS	Correlation Coefficient	.025
		Sig. (2-tailed)	.924
		N	17
	SUBNORM_PEER	Correlation Coefficient	.150
		Sig. (2-tailed)	.566
		N	17
	PBC_RES	Correlation Coefficient	.004
		Sig. (2-tailed)	.989
		N	17

	PBC_SKILL	Correlation Coefficient	.491*
		Sig. (2-tailed)	.045
		N	17

		A.2 I currently use pair programming in my projects	BI
SUBNORM_SS	Pearson Correlation	-.027	.709**
	Sig. (2-tailed)	.918	.001
	N	17	17
SUBNORM_PEER	Pearson Correlation	.389	.398
	Sig. (2-tailed)	.123	.113
	N	17	17
PBC_RES	Pearson Correlation	.300	.538*
	Sig. (2-tailed)	.243	.026
	N	17	17
PBC_SKILL	Pearson Correlation	-.005	.690**
	Sig. (2-tailed)	.983	.002
	N	17	17

		A.2 I currently use pair programming in my projects	BI
ATT	Pearson Correlation	.194	.744**
	Sig. (2-tailed)	.456	.001
	N	17	17
SUBNORM	Pearson Correlation	.382	.632**
	Sig. (2-tailed)	.131	.007
	N	17	17
PBC	Pearson Correlation	.054	.797**
	Sig. (2-tailed)	.838	.000
	N	17	17

Appendix I: Ethical Clearance Approval

UNIVERSITY OF KWAZULU-NATAL
INYUVESI YAKWAZULU-NATALI

02 October 2017

Ms Prashika Dhoodhanath (211503263)
School of Management, IT & Governance
Westville Campus

Dear Ms Dhoodhanath,

Protocol reference number: HSS/1664/017M
Project title: Factors influencing software developers' use of pair programming in an agile software development methodology environment

Approval Notification – Expedited Approval

In response to your application received on 08 September 2017, the Humanities & Social Sciences Research Ethics Committee has considered the abovementioned application and the protocol has been granted **FULL APPROVAL.**

Any alteration/s to the approved research protocol i.e. Questionnaire/Interview Schedule, Informed Consent Form, Title of the Project, Location of the Study, Research Approach and Methods must be reviewed and approved through the amendment/modification prior to its implementation. In case you have further queries, please quote the above reference number.

PLEASE NOTE: Research data should be securely stored in the discipline/department for a period of 5 years.

The ethical clearance certificate is only valid for a period of 3 years from the date of issue. Thereafter Recertification must be applied for on an annual basis.

I take this opportunity of wishing you everything of the best with your study.

Yours faithfully

.....
Dr Shenuka Singh (Chair)

/ms

Cc Supervisor: Ms Rosemary Quilling
Cc Academic Leader Research: Professor Isabel Martins
Cc School Administrator: Ms Angela Pearce

Humanities & Social Sciences Research Ethics Committee
Dr Shenuka Singh (Chair)
Westville Campus, Govan Mbeki Building
Postal Address: Private Bag X54001, Durban 4000
Telephone: +27 (0) 31 260 3587/6350/4557 **Facsimile:** +27 (0) 31 260 4609 **Email:** ximbap@ukzn.ac.za / snymanm@ukzn.ac.za / mohunp@ukzn.ac.za
Website: www.ukzn.ac.za