INVESTIGATIONS ON EXPLORING THE EFFICACY OF DISTRIBUTED PAIR PROGRAMMING IN ACADEMIC ENVIRONMENT

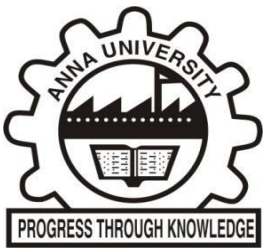
A THESIS

Submitted by

MOHANRAJ N

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY



FACULTY OF SCIENCE AND HUMANITIES

ANNA UNIVERSITY

CHENNAI 600 025

MARCH 2016

ANNA UNIVERSITY
CHENNAI 600 025

CERTIFICATE

The research work embodied in the present Thesis entitled "**INVESTIGATIONS ON EXPLORING THE EFFICACY OF DISTRIBUTED PAIR PROGRAMMING IN ACADEMIC ENVIRONMENT**" has been carried out in the Department of Computer Applications, PSG College of Technology, Coimbatore. The work reported herein is original and does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion or to any other scholar.

I understand the University's policy on plagiarism and declare that the thesis and publications are my own work, except where specifically acknowledged and has not been copied from other sources or been previously submitted for award or assessment.

MOHANRAJ N
RESEARCH SCHOLAR

Dr. A. SANKAR
SUPERVISOR
Associate Professor
Department of Computer Applications
PSG College of Technology
Coimbatore-641 004.

ABSTRACT

In a software development organization, where the traditional way of developing a software program is adopted, a programmer will use a computer terminal for writing software programs, which are necessary for a project. In this approach, the programmer has to consult a systems analyst for implementing the system requirements in his/her software program. This process is time consuming, may lead to miscommunication and thus results in low user satisfaction and low productivity. Also such traditional approaches involve large amounts of documentation like requirements specifications, architecture document, design document and test plans etc., instead of giving out useful functionality to the end user. Due to such user unfriendly approaches sometimes projects are abandoned even before it is deployed.

The solution for such type of problems like project abandonment before deployment, low user satisfaction and low productivity is provided by using Agile software development methods like Pair Programming, Distributed Pair Programming and Extreme Programming. When a single programmer uses the system, many tools are not required for synchronizing the activities, but it is required in the case of Pair Programming or Extreme Programming. The tools such as to replicate a user's desktop onto multiple computers in particular two in the case of pair programming are required. All input and output methods should be shared between multiple computers and the application to be developed should also be deployed on both or multiple computers. It should be noticed that direct communication is better than a detailed documentation. However, documentation is also important in the software development. In pair programming, the limitations are scalability as well as co-located pairs in the same physical location. But, due to the advances of internet and social networking, one can foresee an approach

which uses the advantages of such technologies. Hence, there is a need to address distributed pair programming where there is a possibility of scalability as well as the users need not be co-located in the same physical location.

Distributed pair programming is a practice where two pairs are geographically separated and are working for the same problem. High quality products are produced more rapidly. Remote pair programming, also known as virtual pair programming or distributed pair programming, is a pair programming in which the two programmers are in different locations, working via a collaborative real-time editor, shared desktop or a remote pair programming IDE plugin. But remote pairing has difficulties like extra delays for coordination, loss of verbal communication resulting in confusion and ego conflicts. Software tools may be necessary for screen sharing and for audio chatting through the use of headsets which will be of very useful in distributed pair programming.

In the first part, the effectiveness of the distributed pair programming was analysed using an experimental technique in an academia environment for laboratory courses. This experiment demonstrates that distributed pair programming could be very effective in promoting a student's ability to learn programming concepts in laboratory courses in a faster and more efficient way when compared to solo programming. This experiment was conducted in the computer laboratories in an engineering college. The solutions given by pairs in distributed environment are analysed and shown using various charts. The results provide support for distributed pair programming in the software engineering curriculum of an academia. The results proved that distributed pair programming is better than compared with solo programming in helping a student to learn the programming concepts in the laboratory, easily and efficiently.

Traditionally, pairs are formed based on individual preferences or administrative authority's decision to support organization requirements as there are no standard procedures for forming pairs. A performance will be highly productive, if they are more compatible with each other. Incompatible pairs have less understanding and have communication gap between them. This leads to reduced performance, demotivation and disengagement from work. Then, there is a need for finding compatible pairs. In the second part, a novel method is proposed to form student pairs for programming laboratory courses based on weighted graph matching technique incorporating necessary psychological factors for compatibility between pairs. The concept of graph matching is used in many industrial applications. For instance, the assignment of individual workers to tasks, jobs to processors, etc can be modelled using graph matching. The experimental results demonstrate that the proposed method yields better performance of the pairs in the task assigned to them.

Laboratory courses constitute one of the core competencies that graduates from computer science discipline are expected to possess. Research has suggested that the lack of a formalised structure for laboratory courses may be one of the factors responsible for learners' negative impressions of E-learning. In order to motivate E-learners and present laboratory courses as an easy and attractive challenge, pair programming was used as an effective tool. In the third part, experiments were conducted to analyse whether pair programming can enhance the E-learning system and thereby encourage the E-learners by motivating their E-learning experience. The results show that for the students having no programming background, they gain maximum learning experience from the laboratory work. Most students reported a high learning experience and satisfaction level when E-learning was employed with pair programming.

In the fourth part, a pair recommender system is proposed based on association rule. Association rules are used in data mining to discover interesting relations. In pair programming experiment, association rules are used to discover pair compatibility. Pair compatibility based on previous successful projects, skill levels, designation, personal interests are also identified. Then association rules can be built and also they can be analysed whether they are strongly based on threshold values. To generate strong association rules Apriori algorithm was used. Experimental results show that the proposed approach is better than solo programming and pair programming based on weighted graph approach.

The experiments conducted in the academia environment to evaluate the performance of distributed pair programming proved that distributed pair programming is a better approach than compared with solo programming in improving the ability of student community to learn computer programming without any difficulty and students were also motivated during the distributed pair programming which improved the quality of software products delivered by them. The experiments conducted to find the pair compatibility found to be successful in forming better pairs to produce a better quality software.

ACKNOWLEDGEMENT

I wish to record my deep sense of gratitude and profound thanks to my research supervisor **Dr. A. Sankar,** Associate Professor, Department of Computer Applications, PSG College of Technology, Coimbatore, for his keen interest, inspiring guidance, constant encouragement with my work during all stages, to bring this thesis into fruition.

I would like to thank my Ph.D Doctoral committee members **Dr. P. Narayanasamy** and **Dr. S. Subramanian** for their valuable comments and suggestions.

I wish to thank the Management and our Principal **Dr. R. Rudramoorthy,** PSG College of Technology, Coimbatore for allowing me to use all the facilities in the College without which the work would not have been completed successfully. I would like to thank **Dr. R. Nadarajan,** Professor & Head, Department of Applied Mathematics & Computational Sciences and **Dr. A. Chitra**, Professor & Head, Department of Computer Applications, PSG College of Technology for their valuable support to complete this research work.

I also thank the faculty and non-teaching staff members of the Department of Computer Applications and Department of Applied Mathematics & Computational Sciences, PSG College of Technology, Coimbatore, for their valuable support throughout the course of my research work.

Finally, I thank the Lord Almighty for giving me the knowledge and strength to carry out this research work.

MOHANRAJ N

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.

LIST OFABBREVIATIONS

ADD	-	Agile Dispersed Development
AHP	-	Analytic Hierarchy Process
ANOVA	-	ANalysis of VAriance
AO	-	Agile Outsourcing
ARM	-	Association Rule Mining
COLLECE	-	COLLaborative Edition, Compilation and Execution
COPPER	-	COllaborative Pair Programming EditoR
CRT	-	Cathode Ray Tube
CS	-	Computer Science
DAD	-	Distributed Agile Development
DPP	-	Distributed Pair Programming
DXP	-	Distributed Extreme Programming
EP	-	Extreme Programming
IDE	-	Integrated Development Environment
IT	-	Information Technology
LOC	-	Lines of Code
PG	-	Post Graduate
QA	-	Quality Assurance
TDD	-	Test Driven Development
UG	-	Under Graduate
VLE	-	Virtual Learning Environment
VNC	-	Virtual Networking Computing
XP	-	Extreme Programming

CHAPTER 1

INTRODUCTION

Agile Methodology which is an emerging popular methodology in the software industry as well as in the academia is the core concept of this research work. Extreme Programming is an element of agile methodology and pair programming is one the principles of Extreme Programming (XP) which forms the basis for this research work.

1.1 AGILE METHODOLOGY

Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing and cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement and encourages rapid and flexible responses to change.

The components of agile methodology are given below:

- *Individuals and interactions*: In agile software development, self-organization and motivation are the important factors which will improve the interaction between the pairs in the pair programming environment. New solutions and innovative ideas emerge when the pairs interact with each other. Flaws in old solutions also come to life.

- *Working software*: Working software is the primary measure of the progress in the software development process. Demonstration of working software is considered as the best means of communication between the customer and developer to understand the customer requirements, instead of just depending upon the documentation.
- *Customer collaboration*: As the requirements cannot be fully collected at the beginning of the software development due to various reasons, continuous customer and stakeholder involvement are very important to collect complete requirements of the customer.
- *Responding to change*: Agile methods are focused on quick responses to change and continuous development. Building a plan is useful and each of the agile methodologies contain specific planning activities. They also contain mechanisms for dealing with changing priorities.

Kent Beck, Alistair Cockburn, Martin Fowler, Ron Jeffries and Jim Highsmith (Jim Highsmith 2001) formed the Agile Alliance, a non-profit organization that promotes software development according to the manifesto's values and principles.

1.2 EXTREME PROGRAMMING

The intent of Extreme Programming (XP) is to improve software quality and responsiveness to changing customer requirements. This is done by frequent releases in short development cycles (time boxing) and checkpoints to adopt new customer requirements. According to Ron Jeffries, “Extreme Programming is a discipline for software development, based on

values of simplicity, communication, feedback and courage" (Ron Jeffries et al. 2000).

1.3 XP PROCESS

XP encompasses a set of rules and practices that occur within the context of four framework of activities: planning, design, coding and testing. The methodology takes its name from the idea that the beneficial elements of traditional Software Engineering practices are taken to 'extreme' levels, on the theory if some is good more is better. Process begins with the creation of a set of stories which are called as "user stories".

User stories describe required features and functionality for software to be built. The customer assigns a value to the story based on overall business value of the feature. Members of the XP team then assess each story and assign a cost measured in development weeks, required for the software to be built. New user stories can be written at anytime. If the user story requires more than three development weeks, then the customer is asked to split the user story into smaller size user stories and the assignment of values and cost occurs again. Once a basic commitment is made for the release, the XP team orders the stories that will be developed in one of these ways mentioned below :

- All user stories will be implemented within two to three weeks.
- The user stories with highest value will be moved up in the schedule and implemented first.

As the customer has rated each user story, based upon the business value, the user stories which has highest business value will be assigned with highest priority for implementation.

- The user stories with highest risk, will be moved up in the schedule and implemented first.

User stories, which are very critical for the project and which has the highest risk factor, will be identified by the user and will be assigned with highest priority for implementation.

After the first release (Software increment), the XP team computes the project velocity. Project velocity is the number of user stories implemented during the first release. It is a measure of how much work is getting done on your project. It is a very useful measure for planning the iterations. Project velocity goes up by allowing developers to ask the customers for another story when their work is completed early and no clean up tasks remain.

1.4 XP VALUES

The values which are emphasized by Extreme Programming process are very vital for a successful software project. The values are Communication, Simplicity, Feedback, Courage and Refactoring.

1.4.1 Communication

Effective communication is needed between everyone involved in the project like team members, managers and customers. XP insists on having a real customer working directly with the project. Customer should be able to represent the needs of the fellow users. Customer must be empowered to answer questions and make decisions regarding feature priority, risks, and so forth. They will take part in planning by writing and prioritizing user stories and decide the release content. When the customers, work along with the development team, the development team will get constant feedback from the

customers. The customers can also answer any questions asked by the development team related with the user stories.

1.4.2 Simplicity

Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later. XP approach, focuses on designing and coding for the needs of today instead of those of tomorrow, next week, or next month. This is an advantage, because we are not going to invest our time in all possible future requirements that might change before they become relevant. Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed, while perhaps delaying crucial features. Simplicity in design and coding will improve the quality of communication. A simple design with very simple code could be easily understood by most of the programmers in the development team.

1.4.3 Feedback

Kent Beck has quoted that "Optimism is an occupational hazard of programming. Feedback is the treatment."

Constant feedback is an imperative to make sure that the process is done correctly and to make corrections as soon as possible if there is any defect. Within extreme programming, feedback relates to three dimensions of the system development and are mentioned below.

- Feedback from the system is obtained by testing the programs through unit testing or by exercising periodic integration tests. The number of errors encountered during the unit testing process will reveal the quality of the code. The

programmers will have direct feedback from the state of the system after implementing changes.

- Feedback from the customer is obtained by testing the programs through acceptance testing. The functional tests are conducted jointly by the customer and the tester. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so that the customer can easily know whether his/her requirements are implemented correctly.
- Feedback from the project team is obtained, when customers come up with new requirements in the planning game and the team directly gives an estimation of the time that it will take to implement those requirements.

These three dimensions in the feedback are very important to improve the quality of the software product to be delivered.

1.4.4 Courage

Courage is required for doing XP. Sometimes major refactoring of the system has to be done. There is a need to make big decisions, support them and follow those decisions.

1.4.5 Refactoring

Refactoring is not changing the code by random hacking. It is a disciplined, rigorous approach to improve our code gradually without changing its external behavior.

Code for today and not for tomorrow: Customer requirements keep on changing. So code should be developed focusing on current user requirements and not on future needs.

Refactor as appropriate: Refactoring improves the design of software. Refactoring makes the code easier to understand, even when the code review is done. Refactoring helps to find the defects easily by giving a better perspective on the code. Refactoring speeds up the process because the defects are eliminated.

Be willing to throw code away: Whenever there is a change in the customer requirement, there may be a necessity to change the design and as a result old code must be thrown away. Programmer should have the courage to do this and to do code refactoring.

1.5 EXTREME PROGRAMMING PRACTICES

Extreme Programming (XP) is an agile software methodology to implement software projects. The following practices are used in this methodology.

- Pair Programming
- Test Driven Development
- Continuous Integration
- Coding Standards
- Collective Code Ownership

The practices which are used in this methodology are explained in detail.

1.5.1 Pair Programming

Pair programming (sometimes referred to as peer programming) is an agile software development technique in which two programmers work as a pair together on one workstation. One, the driver, writes code while the other, the observer, pointer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently. There is also analysis, design, refactoring, testing and code review in addition to programming. Sufficient space should be provided so that two persons can sit comfortably at a single computer, see the computer, see the monitor and share the keyboard and mouse effectively. One of the persons in the pair programming exercise is called as the driver and other one who is going to guide the driver is called as a navigator. The driver has the control of the keyboard and mouse and he/she is involved actively in programming. The partner watches, offers advice, makes suggestions, points out mistakes, questions decisions and generally works as a back-seat driver keeping an eye on the strategic goals while the driver concentrates on the tactical details. Roles can change often. In Figure 1.1, pair programmers are involved in a software development activity.

The various activities that occur during pair programming process are given below:

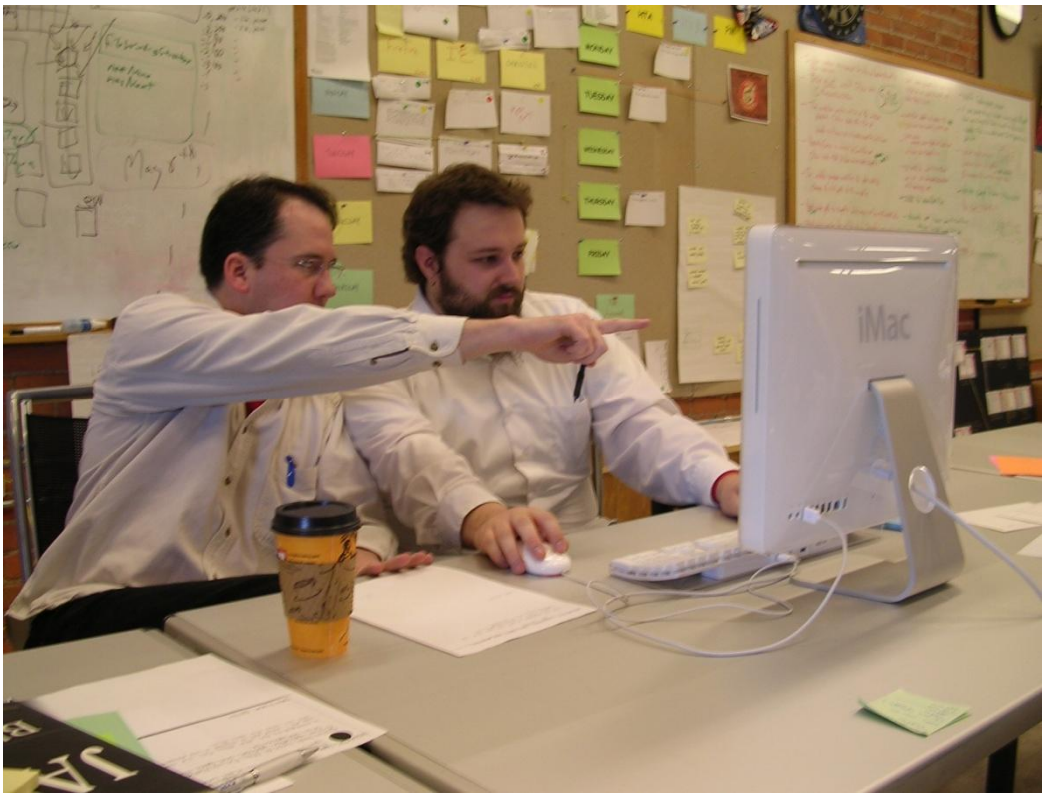
- Making design decisions
- Implementing code
- Reviewing code
- Testing code

- Refactoring code
- Communicating with the partner and indirectly with the other team members too.
- Educating each other and again indirectly the whole team

Dr. Laurie Williams, Department of Computer Science, North Carolina State University has explained the concept of pair programming (Williams 2002). In this approach, two programmers who are seated side-by-side will be working and collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively involved in the program development. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together and make equal contribution in the software development.

1.5.1.1 Remote Pair Programming

Remote pair programming or distributed pair programming, is pair programming in which the two programmers are seated in different locations, working in a computer system for the same product. Communication is established between the two programmers by means of software tools that will enable audio and video chatting and for sharing the desktop. Remote pairing may have difficulties that is not present in face-to-face pairing, like extra delays for coordination among pairs, depending more on "heavyweight" task-tracking tools instead of "lightweight" ones like index cards and loss of verbal communication resulting in confusion and conflicts over such things as who "has the keyboard".



(Source: https://en.wikipedia.org/wiki/Pair_programming)

Figure 1.1 Pair Programming at Software Industries

1.5.1.2 Advantages of pair programming

Quality and productivity are improved by following the pair programming process. The advantages of pair programming process are given below:

- Economics

It is much economical to use pair programming, because it reduces the expenses by reducing the defects in the programs. Pairs spend about 15% more time on programs than individuals. However, the resulting code has about 15% fewer defects. Along with code development time, other factors like field support costs and quality assurance also affect the expenses. IBM

reported spending about “$250 million repairing and reinstalling fixes to 30,000 customer-reported problems”. Pair programming significantly reduces these expenses by reducing the defects in the programs (Alistair Cockburn & Laurie Williams 2001).

- Design quality

Design quality is improved to a large extent by adopting the pair programming. A system with two programmers possesses greater potential for the generation of more diversified solutions to problems for the following three reasons:

i) The programmers bring different prior experiences to the task;

ii) They may access information relevant to the task in different ways;

iii) They stand in different relationships to the problem by virtue of their functional roles.

In the attempt to share goals and plans, the programmers must overtly negotiate a shared course of action when a conflict arises between them. In doing so, they consider a larger number of ways of solving the problem than a single programmer alone might do. This significantly improves the design quality of the program as it reduces the chances of selecting a poor method.

- Satisfaction

Satisfaction is one of the motivating factors for the pair programmers which results in quality and productivity. In an online survey of pair programmers, 96% of them stated that they enjoyed their work more than when they programmed alone. Additionally, 95% of the surveyed

programmers stated that they were more confident in their solutions when they are involved in pair programming (Williams et al. 2003). A correlation exists between satisfaction among programmers and their confidence in the code building i.e. the pairs enjoy their work more because they are more confident in it.

- Learning

Knowledge is constantly shared between pair programmers, from tips on programming language rules to overall design skills. In "promiscuous pairing", each programmer communicates and works with all the other programmers on the team rather than pairing only with one partner, which causes knowledge of the system to spread throughout the whole team. Pair programming allows the programmers to examine their partner's code and provide feedback which is necessary to increase their own ability to develop monitoring mechanisms for their own learning activities.

- Team-building and communication

Pair programming allows team members to share problems and solutions quickly making them less likely to have hidden agenda from each other. This helps pair programmers to learn to communicate more easily. If the pair can work together, then, they learn ways to communicate more easily and they communicate more often. “This raises the communication bandwidth and frequency within the project, increasing overall information flow within the team” (Alistair Cockburn & Laurie Williams 2001).

In Figure 1.2, one can see the pair programmers are involved in various activities of software development in an academic institution.

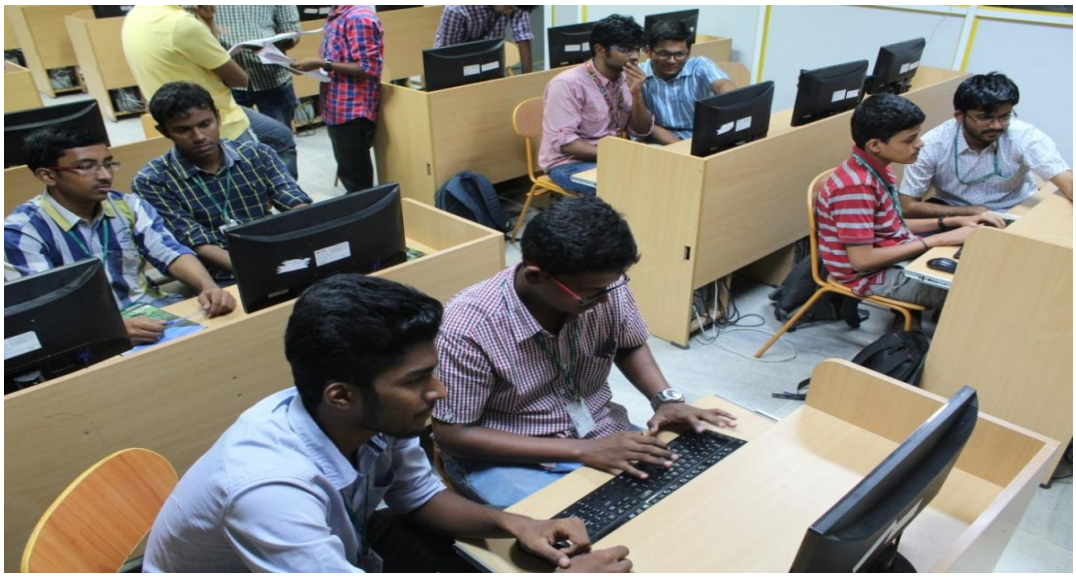


Figure 1.2 Pair Programmers at their work in academia

1.5.2 Test Driven Development

Test Driven Development (TDD) is a technique in which the test cases are prepared before writing the code which is going to be tested. This technique results in developing a code with less number of defects. Because the test cases are prepared first, the code should be written in such a way that it can be tested thoroughly. Preparing of test cases at the initial stage of code development, encourages a programmer to write simple and single-purpose method. Because the methods will be called from more than one environment, they tend to be more independent of the environment.

Test cases should run at a 100% pass rate in the production code. If a test case fails during integration testing, the newly added code may contain some errors. To avoid this problem, regression testing should be carried out whenever a new code is added.

Regression testing is a type of software testing that verifies that software that was previously developed and tested still performs correctly after it was modified or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc.

1.5.3 Continuous Integration

Continuous Integration is the process of developing a module in such a way that the module can be integrated together with other existing modules, as soon as the module is developed and this integration is not done at the final stage during integration testing. Unit testing is testing a single, independent module of a software system in isolation. Integration testing is testing the complete system with all its modules integrated together, to make certain that all the modules work properly when they are integrated. Continuous integration is performing integration tests frequently. Continuous integration is more important for larger and complex software projects.

1.5.4 Simple Design

The policy behind this practice is that the simple design can possibly work. Complex design is almost and always a very bad investment. Complexity to support future features is seldom a good idea.

Programs should be focused on features that are required today and not for the requirements required in future. Refactoring the code for additional tests can be done easily with the help of efficient unit test cases.

1.5.5 Coding Standards

Coding standards are important for many reasons. First and foremost, they specify a common format for the source code and comments. This allows developers to easily share code, and the ideas expressed within the code and comments, between each other.

Coding standards helps to keep the code consistent and easy for the entire project team to maintain and refactor. Different programmers in a team, use the common coding standard for writing their code and thereby collective

code ownership is established. Coding standards can be easily achieved by the pair programmers. More importantly, a well designed standard will also detail how certain code should be written, not just how it looks on screen.

1.5.6 Collective Code Ownership

Ownership should be taken for the partner's code during the pair programming. XP team practices collective code ownership. Anyone can work on any part of the system at any time. There are no class owners. There is no need to request for changes to classes outside the jurisdiction. If any change is to be done or a method is to be added, that can be done easily. Collective ownership enables coding with intention because functionalities can be added wherever and whenever needed. If needed, complex code can be made simple and refactoring can be done without much difficulty.

1.6 OBJECTIVE OF THE DISSERTATION

Software development and maintenance activities are becoming more complex and quality of software products is very much essential for the successful completion of a project. Projects have to be completed within the estimated time, budget and with expected quality. Traditional methodologies are not so useful in achieving this. New methodologies have to be exercised for the successful project outcomes.

Many methodologies have been proposed for software development. One of the new methodologies is the Distributed Pair Programming (DPP) approach, which needs to be analyzed for a successful outcome. Quality and Productivity has to be measured out of this new methodology.

The objective of this thesis is to analyse the efficiency of distributed pair programming practices by measuring and comparing the productivity and quality with that of the traditional programming practices

and to develop a strategy for successful pair matching which is one of the key issues in distributed pair programming. The main contributions of the thesis are:

- Investigations and Analysis of the results of the experiments conducted for finding the efficiency of distributed pair programming.
- A novel method proposed for successful pair matching and effectiveness of that method is analysed because pair matching is one of the essentials for successful pair programming activity.
- An investigation and analysis for enhancing learning experience of e learners in laboratory courses.
- A pair recommender system proposed to discover pair compatibility, based on association rule which is used in data mining to discover interesting relations.

1.7 THESIS ORGANIZATION

The objective and motivation for agile methodology is discussed in this chapter. The objectives and values of Extreme Programming are also discussed here. Pair Programming and Distributed Pair Programming, the key practices of Extreme Programming are described in this chapter. The remaining chapters of this thesis are organized as mentioned below:

In **Chapter 2**, literature survey and results related with the efficiency of pair programming and distributed pair programming are analyzed and results are discussed and summarized.

In **Chapter 3**, the results of experiments conducted to evaluate the efficiency of distributed pair programming, compared with solo programming are investigated and analyzed. Twenty metrics were selected for evaluation purpose. A software tool was developed for communication between remote pair programmers. The experimental results show that Distributed Pair Programming yields best quality and productivity than compared with that of solo programming.

In **Chapter 4**, a novel method is proposed and discussed to form student pairs for programming laboratory courses based on weighted graph matching technique, incorporating necessary psychological factors for compatibility between pairs. Analysis of experimental results show that the proposed method for pair matching using weighted graph matching technique is an efficient way for pair matching which will help in successful pair compatibility and thereby yield the productivity and quality which are necessary for a successful outcomes.

In **Chapter 5**, a method for enhancing the learning process of e-learners in a lab assignment is suggested. Various experiments were conducted and the results suggest that both pair programming and distributed pair programming can enhance the learning experience of e-learners.

In **Chapter 6**, a Pair Recommender system is proposed based on Association Rule. Association rules are used in data mining to discover interesting relations. In pair programming, association rules are used to discover pair compatibility. Pair compatibility based on previous successful projects, skill levels, designation, personal interests are also identified.

In **Chapter 7**, conclusion of the entire research work are presented, the summary of contributions for the thesis is given and the scope for future research opportunities is also mentioned in this chapter.

CHAPTER 2

LITERATURE SURVEY

Various research works are going on across the globe to analyse the efficiency of distributed pair programming in software industry and in academia. Those published research findings were surveyed in this chapter, analyzed and summarized.

2.1 SOFTWARE DEVELOPMENT PRACTICE

Agile management or Agile process management or simply agile refer to an iterative, incremental method of managing the design and build activities for engineering, information technology, and other business areas that aim to provide new product development in a highly flexible and interactive manner.

Agile programming involves delivering the software which is working efficiently, after a thorough testing. This work is carried out as several iterations with a time duration of two to four weeks. As the iterations are created and implemented, work pressure also increases as the customer requirements keep on increasing. Under such pressures, when there is a traditional programming practice, which will have more time consuming analysis, design and testing phases, large amounts of documentation has to be produced which is a time consuming process. As a result of this, software will be delivered late. Fortunately, the best practices of agile have proven to enable more frequent delivery of products with higher quality. These agile best practices help the programmers and hence the code itself become more

agile. The smaller cycles of agile programming appear to be less rigorous, but the effectiveness comes from the application of these practices with great discipline. This discipline leads to extensible, code with less defects and robust design that will work efficiently. It is well-factored and well-protected by unit tests.

There is significant dependence on personal communication and customer collaboration. Agile methodology can be difficult to apply in the following situations:

- When the team size is large and the team members are without adequate software tools required for software development.
- When the team members are unable to share their ideas and there is difficulty to communicate with each other.
- When the team members have no exposure to agile methodology.

By analyzing the problems and difficulties experienced by team members in each of the situations, solutions can be found by conducting research experiments for an effective implementation of agile methodology.

2.1 AGILE PROGRAMMING ENVIRONMENTS

A problem arises to maintain close collaboration practices and run agile project in a distributed setup (Concas et al. 2007). As a solution to this problem, a suitable tool support is usually employed; however, it seems insufficient at the moment. (Concas et al. 2007) present a set of general requirements that become a basis for further investigation into distributed collaboration needs and challenges. As a verification of initial assumptions, a new system was designed and part of it, which is responsible for supporting

distributed pair programmers, implemented and experimentally evaluated. The first group includes conferencing applications like Microsoft NetMeeting, virtual whiteboards and desktop sharing solutions. The second group tool is TUKAN environment with a pair programming oriented tool consisting of voice-video connection and other communication means. It proposes the following general requirements :

- The system must support (preserve, stimulate, not suppress) the phenomenon of synergy which is not only the most valuable but also crucial factor, especially under the circumstances of a team and distribution of its pairs.
- The system ought to cover all functions that are recognized as necessary or useful in the geographically co-located mode, which stay in accordance with the primary requirement, including functions which are decisive only for the friendliness of it.
- The system must fulfill all requirements for a modern computer system of its type as long as a conflict with the primary or secondary requirements does not arise.

The editor was developed (Concas et al. 2007) as a part of a larger system called Agile Studio, meant for supporting selected agile practices. It has been observed that every collaboration is likely to take advantage of certain shared objects. Therefore, the editor is based on server-client architecture, where server side is responsible for sharing synchronized instances of the session objects through source files.

Three general cases for non-colocated, agile aligned development (Alistair Cockburn & Laurie Williams 2001) are as mentioned below:

Agile Outsourcing (AO) is a concept where an agile team is created at an appropriately low cost offshore location. Requirements are given by onshore team using shared documents. Here onshore team refers to the team which may be in another country or a different location.

Agile Dispersed Development (ADD) is practiced by many of the Open Source community and also by few commercial companies.

Distributed Agile Development (DAD) is an approach where customers are distributed. One development team is distributed evenly over several sites to remain close to the customers.

Team members who are involved in Distributed Extreme Programming (DXP) as well as Distributed Pair Programming (DPP) are provided with as many communication media as possible (Alistair Cockburn & Laurie Williams 2001). At least communication media like individual, conference telephone, teleconference, video conference, email, IM, wiki and VNC will be provided. Widely separated team members need to maintain a common identity as solution providers for the technical problem. They need to share rights and responsibilities toward each others' work, just as co-located workers do.

Members of a team in one location find it difficult to understand the point of view of members from another location. Trust and cooperation break down amongst the members; it is hard for one local group to work effectively with another. Team members find it hard to have faith in the good intentions of remote colleagues. Blamestorming replaces collaboration; finger pointing replaces problem solving (Alistair Cockburn & Laurie Williams 2001).

The following Agile principles allow development teams to do their work efficiently.

- Distributed Standup
- Multiple Communication Modes
- Remote Pair
- One Team, One Codebase
- Functional Tests Capture Requirements
- One Team, One Build
- Code is Communication
- Tests Announce Intention

Convention speaks against having two people work together to develop code – having “two do the work of one”, as some people see it. Managers view programmers as a scarce resource and are reluctant to "waste" such by doubling the number of people needed to develop a piece of code and also experienced programmers are very reluctant to program with another person. Some say their code is "personal," or that another person would only slow them down. Others say working with a partner will cause trouble coordinating work times or code versions. But it must be noticed as several well-respected programmers prefer working in pairs, making it their preferred programming style. Seasoned pair programmers describe working in pairs as "more than twice as fast”. Qualitative evidence suggests the resulting design is better, resulting in simpler code, easier to extend. Even relative novices contribute to an expert' programming, according to interviews.

2.2 EXTREME PROGRAMMING ENVIRONMENTS

DXP (Distributed Extreme Programming) and open source processes can be used as a baseline according to Wells & Williams (2002) and in that case, the work processes of virtual software teams are improved.

XP teams are usually much more closely coordinated than open source projects. Hence, project coordination support is strongly required for DXP. Here the tasks are assigned to XP team in a coordinated manner and deadlines are set as well as overview of the current state of the project is also updated. Team members access their to-do lists and to perform their tasks they retrieve relevant information in a coordinated way.

To establish synchronous communication, extensive e-mails are used as well as audio and video calls and text chat are also used. In the case of pair programming sharing of their application is also done. Both pull as well as push access of information is done for the user.

The MILOS framework discussed in (Wells & Williams 2002) nicely fits the requirements on DXP support. The overall goal of the MILOS approach is to support process execution and organizational learning for virtual software development teams. In this section , how MILOS supports Distributed XP (DXP) is explained. The support provided by MILOS should be minimally intrusive to reduce overhead: MILOS stands for "**M**inimally **I**nvasive **L**ongterm **O**rganizational **S**upport". The MILOS approach can be applied for open source projects as well as for commercial teams that are distributed over the world. It was adapted to support distributed XP.

Nevertheless, several extensions for supporting distributed XP like user stories in which a new product type that represents user stories was added. In addition, whenever a new user story is entered, MILOS ASE automatically adds a task for implementing this story into the task list. Also release and iteration planning allows easily defining and changing releases, iterations, user stories, and tasks. In a distributed setting, the system provides awareness on what is going on in the project based on four task levels from XP namely, release, iteration, user story and task. Further MS NetMeeting is integrated to be able to support distributed pair programming and synchronous communication.

2.4 PAIR PROGRAMMING ENVIRONMENTS

Pair programming is one of the twelve practices of Extreme Programming (XP) (Beck 2000). In pair programming it is assumed that the pairs will be working in front of the same workstation (Wells & Williams 2002). If Extreme Programming is to be used for distributed development of software, collocation becomes a limitation. A variant of Extreme Programming is used through distributed pair programming or virtual teaming.

A Virtual team can be defined as a group of people who work together towards a common goal, but across time, distance, culture and organizational boundaries (George & Mansour 2002).

(Source : https://github.com/FreeCodeCamp)

Figure 2.1 Typical Pair Programming Environment

Pair programming (Brian 2004) transforms what has traditionally been a solitary activity into a cooperative effort. One of the developers, called the driver, controls the computer keyboard and mouse. The driver is responsible for entering software design, source code, and test cases. The second developer, called the navigator, examines the driver's work, offering advice, suggesting corrections, and assisting with design decisions. The developers switch roles at regular intervals. Figure 2.1 shows the typical pair programming environment. Although role switching is informal, a typical interval is 20 minutes in between role switching.

The experiment conducted at North Carolina State University, United States of America, is a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects (Wells & Williams 2002). The following were observed based on the experiment :

- Distributed pair programming in virtual teams is a feasible way of developing software. Virtual teams represent the teams which are geographically separated.
- Software development involving distributed pair programming is comparable to that developed using co-located pair programming or virtual teams without distributed pair programming.
- The two metrics used for this comparison were productivity (in terms of lines of code per hour) and quality (in terms of the grades obtained) while conducting laboratory experiments.
- Co-located teams did not achieve statistically and significantly better results than the distributed teams.

- Feedback from the students indicate that distributed pair programming fosters teamwork and communication within a virtual team.

The features expected out of a typical distributed pair programming tool (Hiroshi 2003) are as given below.

- Synchronous editing of source code: As is the case for any modern source code editor it should highlight keywords based on the programming language being used and provide conventional editing tools such as: Cut, Copy, Paste, Find, and Replace.
- Only two programmers need to collaborate at the same time.
- The system should support the options of compiling and executing the source code being edited and should notify the users of the error messages reported by the compiler.
- The source code files to be shared should be stored in Web repositories to ensure that documents are available to all members of the development team. Furthermore, configuration control tools are increasingly being developed on top of Web servers to take advantage of the Web's ubiquity and open standards.
- Access to documents being edited should be controlled at the repository level. Mechanisms to request and obtain shared resources need to be provided.
- Pair programming demands frequent communication between colleagues. The system should support text and audio-based communication.

- Awareness of the presence and state of authors and documents, as well as access rights pertaining to shared resources should be provided to the users.

Data were analyzed in terms of productivity and quality, as defined above. Also, student feedback formed an important input for the experiment. The objective was not to show that, distributed pair programming is superior to co-located pair programming for student teams. But the real objective was to demonstrate that distributed pairing is a viable and desirable alternative for use with student teams, particularly for students who are learning in the distance mode. The results show that distributed teams had a slightly higher productivity as compared to co-located teams; It is to be noted that pair programming:

- It should significantly reduce the risk of subtle errors that would make debugging excruciating;
- It could give us a much broader code review;
- It would provide an opportunity to share knowledge between programmers.

The significant benefits of distributed pair programming are that (Alistair Cockburn)

- Quality: Two developers produce code with less defects. The navigator continuously reviews the code and design, which enables early defect detection and removal during the pairing session.
- Time: Two developers use less time to produce the same quantity of code when compared with a single developer.

- Teamwork: Pair programming can improve the relationship between developers, which builds trust and improves teamwork.
- Knowledge transfer & Learning: Pair programmers can share and learn knowledge from each other during the pairing session.
- Job satisfaction: The intensive collaboration helps job retention for employees

Alistair Cockburn it is found that for a development-time cost of about 15%, pair programming improves design quality, reduces defects, reduces staffing risk, enhances technical skills, improves team communications and is considered more enjoyable at statistically significant levels. It took only 15% more time for the pair programmers than compared with solo programmers for completing the experiment. Significantly the resulting code had 15% less defects for pair programmers when compared with solo programmers.

2.5 DISTRIBUTED PAIR PROGRAMMING ENVIRONMENTS

Schumer and Schumer (Till & Jan 2001) and Maurer (Frank Maurer 2002) have conducted research in this domain and suggest that distributed pair programming (DPP) can work efficiently. In a work by Baheti et al. (Prashant Baheti) suggests that distributed pairing can be as effective as co-located pairing. Canfora et al. (Gerardo et al. 2003) analysed virtual pairing with the help of students, who were using a screen sharing application along with a text-based chat application. No audio channel was provided to the students.

Stotts et al. (David Stotts et al. 2003) provides further evidence of the potential success of distributed pairing. They describe an on-going series of experiments and case studies in which students virtually paired. Although distributed pairs successfully completed their programming assignments, they complained of their inability to point or gesture. As Stotts observed, "pairs need better capabilities for indicating areas of interest".

A representative sample of responses from the pairs where they enjoyed distributed pair programming (Brian 2004) include:

- It allows the pairs to work comfortably of their work place without ever getting out of our chairs and it also helped to overcome some schedule conflicts and the time that would have been wasted just walking to the other person's computer. That time was instead turned into productive time for programming.
- One do not have to go all the way to a computer lab for pair programming.
- It made distributed pair programming very easy and convenient. There was no necessity to meet the pair in the campus or at each other's houses so that distributed pair programming can be done without the effort of getting together. The class would have required a lot more time without the tool.

The following are the responses collected from the students, who have used the distributed pair programming tool.

- Without meeting in person, peers were able to work perfectly. They could work on it any time and also could take long breaks.

- The pair programming tool allowed the peers to work together from two different locations. The pointing function of the program also made it easier to point out errors and did not allow them to do syntax errors while typing the program.
- There is a flexibility in the case of two partners who cannot meet in person and work together. One of the pairs liked being able to work on a separate computer.
- It is easier to work individually in a separate computer, than sharing a computer.
- Being able to switch driver/navigator role easily.

Some of the students did not like the experiment for the reasons given below:

- Communication with the partner using the software tool was not so comfortable as expected.
- The tool was difficult to use when the pairs were programming something they had never programmed it before – for instance, when they tried new data structures for the first time.
- The software tool was not a good way to communicate, even with the headsets. Sometimes it was difficult to explain some concept through the headset.

In the COPPER (COllaborative Pair Programming EditoR) System (Hiroshi 2003), a synchronous source code editor that allows two co-located software engineers to write a program using pair programming. COPPER implements characteristics of groupware systems such as communication mechanisms, collaboration awareness, concurrency control and a radar view of the documents. It also incorporates a document presence module, which

extends the functionality of instant messaging systems to allow users to register documents from a web server and interact with them in a similar fashion as they do with a colleague. The results obtained from a preliminary evaluation report of COPPER, provides the evidence that the system could successfully support distributed pair programming. The audio module establishes and maintains an audio communication channel between two clients so that their users can hold a conversation while collaborating.

Agile methodologies stress the need for close physical proximity of team members. However, circumstances may prevent a team from working in close physical proximity. For example, a company or a project may have development teams physically distributed in different locations. As a result, increasingly many companies are looking at adapting agile methodologies for use in a distributed environment (Wells & Williams 2002).

Prashant Baheti et al. describes the development and study of a technique tailored for distributed programming teams. The technique is based on an emerging software engineering methodology known as pair programming combined with nearly 20 years of widespread and active research in collaborative software systems. Students use interactive information technology over the Internet, such as *PCAnywhere* and *NetMeeting,* to jointly and simultaneously control a programming session and to speak with each other synchronously. The earliest example of a collaborative computer system was NLS-Augment by Engelbart (Engelbart& English 1968), an initial version of which was demonstrated in the early 1960. Engelbart's system used shared CRTs, audio connections, mouse and keyboard to allow crude teleconferencing and shared examination of text files by users who were not co-located. From these early beginnings, collaborative software systems became the subject of widespread research more than 15 years ago, with the creation of the personal computers. Ongoing research

tends to focus in three main areas: hardware to provide effective communications; software that allow sharing of artifacts; and conceptual models of how people want to or are able to interact effectively. The success of the simple DXP platform has led to construct one that collaborators with a more significant video image, including the ability to create hyperlinks in a real-time video stream.

The following hypotheses were examined by (Prashant Baheti et al. 2002) for analyzing the effectiveness of distributed pair programming:

Distributed teams whose members pair synchronously with each other will produce higher quality code than distributed teams that do not pair synchronously. In the academic environment, quality can be assessed by the grades obtained by the students for their project. A statistical T-test can be performed to find whether one of the groups gets statistically and significantly better results at different levels of significance ($p < 0.01$, 0.05, 0.1 etc.).

1. Distributed teams whose members pair synchronously will be more productive than distributed teams that do not pair synchronously in terms of number of lines of code produced per hour.
2. Distributed teams who pair synchronously will have comparable productivity and quality when compared with collocated teams.
3. Distributed teams who pair synchronously will have better communication within the team when compared with distributed teams that do not pair synchronously.
4. Distributed teams who pair synchronously will have better teamwork within the team when compared with distributed teams that do not pair synchronously.

Five out of six (83%) students involved in distributed pair programming thought that technology was not much of a hindrance in collaborative programming. Also, about the same percentage (82%) of students involved in virtual teaming with or without pair programming felt that there was a nice cooperation among team members. The experiment conducted for this purpose was a classroom experiment among 132 students, including 34 distance-learning students. To draw a significant conclusion, such experiments have to be repeated on a larger scale if possible. However, these experiments have given initial indications of the viability of distributed pair programming.

The statistical analysis showed a phenomenon, called the dismissal hypothesis: distributed pairs tend to stop collaboration and begin working as solo programmer (Gerardo 2003). Further it shows that in distributed pair programming, people need a communication media that owns at least two features: vocal communication and a blackboard.

Four causes have been recognized: the faulty phone cause, the stranger cause, the two-minds cause and the anarchy cause. (Canfora et al. 2003)

- A defective communication is one of the four causes of the pair dismissal which is identified as the faulty phone cause.
- When the pairs competence levels are different the pairs cannot be compatible with each other and this is identified as stranger cause. The pair has to present very comparable levels of competence. A way to obtain such a condition is to make the pairs work together in many projects. In this way, it is also possible to prevent the stranger cause.
- When the pairs have difference of opinion, it results in incompatibility between pairs. This is identified as two-minds

cause. The meeting should be realized with closing assessment aiming at verifying that the pair has formed an unique mind. The unique mind is intended as a uniform vision of the domain, strategies, goals, and the overall knowledge to be applied during the project. This should avoid the two-minds cause.

- The pairs should be selected in such a way that they are compatible with each other with mutual interests and without any ego conflicts. This will avoid the anarchy cause.

To manage distributed pair programming efficiently and successfully the following practices have to be adopted:

- Establish a behavioral protocol that defines clearly the roles in the pair and the switching of roles;
- Pairs are to be selected with comparable experience and capabilities;
- Make pairs familiar in working with each other;
- Plan frequent brainstorming in order to create a common vision and goal for the project.

The following hypotheses were considered when comparing the distributed team which paired synchronously and the distributed team which did not pair synchronously by (Prashant Baheti et al. 2002).

- **Hypothesis 1:** Distributed teams whose member's pair synchronously with each other will produce higher quality code than distributed teams that do not pair synchronously.

- **Hypothesis 2:** Distributed teams whose members pair synchronously will be more productive than distributed teams that do not pair synchronously.
- **Hypothesis 3:** Distributed teams who pair synchronously will have comparable productivity and quality when compared with co-located teams.
- **Hypothesis 4:** Distributed teams who pair synchronously will have better communication and teamwork within the team when compared with distributed teams that do not pair synchronously.

2.6 TOOLS FOR AGILE DEVELOPMENT PROCESS

The tool described in (Brian 2004) is based on the open source screen sharing application Virtual Network Computing (VNC) (Tristan et al. 1998). Experiments were conducted (Brian 2004) with control group as well as experiment group with students. Students in the control and experimental groups were performed equally well on the final examination. Although, it is not statistically significant, students who used the tool performed better in the examination than the students in the control group. Students in both experimental groups were also equally confident in their programming solutions. The following comments from the students were collected and analysed.

When some of the students were asked why they were not using the distributed pairing tool, they answered that, they never had any necessity for using the tool, because it was easier for them to meet their peer in the lab and communicate with them.

It is observed that the above remark were given by the students when VNC was used in the same lab and hence the students find meeting peers is easier than using the tool. But when it is required in projects where the users do not meet because of distance between them, then DPP (Distributed pair programming) is necessary.

The COLLECE (COLLaborative Edition, Compilation and Execution) system (Bravo et al. 2007) is a groupware tool that enables users who are located in different workstations to collaborate in the same time (real time) in the building of a software product. COLLECE was used in a study to compare the experiences of Distributed Pair Programmers (DPPs) (Williams & Kessler 2002) and solo programmers. Here in the study mainly the productivity and program quality are considered. It was observed that when the distributed pair programmers have enough experience in the use of the groupware tool and work collaboratively with their partner, the quality of program is better than of those built by solo programmers. But DPPs spent more time in completing their tasks, because they have to carry out additional interactions in order to coordinate and communicate in a distributed collaborative synchronous environment.

2.7 CONCLUSION

Pair programming which is a part of Agile software development method has been one of the leading research areas. Mostly such research setup are academic environment where both the programmers in the pair co-located. This will not be the case when we experiment in real time programmers in the industry. Hence the need for attempting distributed pair programming arises. In this chapter, the recently published research findings related with distributed pair programming were surveyed, analyzed and summarized. The main objective is to find the reasons for incompatibility between pairs where sharing the knowledge with the partner is not effective, which results in low quality.

CHAPTER 3

ASSESSING THE EFFECTIVENESS OF DISTRIBUTED

PAIR PROGRAMMING FOR LABORATORY COURSES

The effectiveness of distributed pair programming was analysed by using an experimental technique for laboratory courses in the academia. The experimental technique is discussed in this chapter.

3.1 DISTRIBUTED PAIR PROGRAMMING

Pair programming is an agile software development technique in which two programmers work together at one computer on the same design, algorithm, code or testing activity. One of the key requirements of pair programming is the strong and effective communication between the pairs. To enable this strong level of communication between the pairs, XP emphasizes that pairs should be physically located close to each other. For various reasons this co-location may not be feasible. The reasons may be due to on-shore projects the pairs may be geographically separated or the pairs may be working in a different locations due to the nature of the project.

Distributed pair programming is the practice where two pairs are geographically separated and are working for a same problem. Higher quality products are produced more rapidly. This chapter demonstrates that distributed pair programming could be very effective in promoting a student's ability to learn programming concepts in lab courses in a faster and more

efficient way when compared to solo-programming. Supportive results are projected to validate the claim using various parameters.

3.2 EXPERIMENTAL SETUP

Subjective affirmation from software companies recommend that software developers are often required to work collaboratively all through their professional life (Nagappan et al. 2003), consequently companies anticipate that future software developers be given such training throughout their undergraduate education (Cliburn 2003). Pair programming has, over the last five years, been look into as a promising technique to offer such skills, apart from other cited benefits. Williams et al. (Williams et al. 2003) describe pair programming (PP) as two programmers jointly produce one artifact. The artifact may be a design or an algorithm or a code. The two programmers are like a unified, intelligent organism working with single mind, responsible for every aspect of this artifact. One of the pair, called the driver, will type the code in the computer or will produce a design document. The other partner, called the navigator, has many roles. One of the roles of the navigator is to observe the work of the driver, looking for defects in his/her work. He/she has to guide the driver in the right direction by giving suggestions for completing the assignment successfully.

The benefits of pair programming are described to include enhanced quality, collaboration, communication, confidence and understanding (Beva et al. 2002; Declue 2003) (Hanks et al. 2004). However, disadvantages have also been suggested, including pair incompatibility (Bevan et al. 2002;McDowell et al. 2003) and unequal participation (Thomas et al. 2003). Mainly studies on pair programming examine subject's collaboration sharing the same computer/desk/paper. However in recent times other studies also looked into the impact of distribution to the effectiveness of pair programming (Canfora et al. 2003; Natsu et al. 2003). In addition, a

number of pair programming trials have been engaged at preliminary courses, and a few at more higher computer science courses.

Only one of the studies in the literature review has used second year students as subjects of a pair programming trial. This chapter presents the results of a distributed pair programming trial conducted in a college level with first year students. This contribution is consequently to add pragmatic confirmation to the present body of familiarity on distributed pair programming regarding this methods worthiness or not for getting better academic results and satisfaction of students.

Similar issues were explored to those reported in (Williams et al. 2003), and in particular at whether distributed pair programming enhance learning and the enjoyment of those students who take part in collaborative activities. A subset of the questionnaires were employed in the experiment which was used by (Abdullah Mohd ZIN et al.2006) while conducting the experiments. Sections 3.1 and 3.2 provides a summary of the distributed pair programming related work, followed by section 3.3 where our experiment is described. In section 3.4, inferences from the metrics analysis to measure the effectiveness of DPP are described. In section 3.5, inferences from the lines of code analysis are described. Inferences from the defects analysis are described in section 3.6 At last, conclusions are given in Section 3.7.

3.3 RELATED WORK

A variety of techniques have been implemented for forming pairs. (McDowell et. al 2002), to permit students to decide their own partners, alike trend observed in (Williams et al. 2000). Nagappan et. al (2003), used a software program to create random partner assignments. Thomas et al. (2003) had students rate their programming abilities and allocated partners based on these ratings. Cliburn (Williams et al. 2003) assigned pairs by consortium of

students from different cultural backgrounds. Nagappan et al. (Nagappan et al. 2003) also discuss forming pairs based on personality profiles such as the Myer-Briggs personality tests and Katira et al. (Katira et al. 2004) suggest that pair compatibility in basic courses may increase if pairs are formed by joining students of dissimilar personality type.

In numerous studies (Cliburn 2003; Declue 2003; Nagappan et al. 2003; Thomas et al. 2003) programmers were assigned to new partners throughout the semester. However, despite evidence of the benefits of pair rotation (Srikanth et al. 2004) much research has not been done related with pair rotation (Bevan et al. 2002; McDowell et al. 2002; Williams et al. 2000). In terms of experimental design five different choices have been identified: i) comparison between paired and solo students using separate groups and over the same semester/sessions (Gebringer 2003; McDowell et al. 2002; McDowell et al. 2003) ii) comparison between paired and solo using same group of students over the same semester/session (Declue 2003) iii) alternating between paired and solo students over several terms/sessions (Hanks et al. 2004) iv) use of only paired students (Cliburn 2003;Williams et al. 2000). Some studies (Williams et al. 2003) have kept the experimental unit fixed (Hanks et al. 2004) while others provided additional or different assignments to paired students.

3.3.1 Experiments conducted to assess DPP

The experiments were conducted with the students of first year of Five Year Integrated M.Sc(Software Engineering). Students of Five Year Integrated M.Sc(Software Engineering) programme are selected for admission based on their outstanding academic performance at their school level and an interview which should reflect their confidence level. The experiments were performed in two ways. In the first way, each student is expected to do a programming assignment as a solo programmer related to a

computer laboratory course. In the second way, the students are expected to do programming assignment as a pair programming and they will be located in two different buildings of the college campus. In the second method, each programmer was given a computer with all the required software so that they can communicate with each other through voice chat, desktop sharing and file transfer options.

Experiment 1 : Solo programming

In this experiment, 36 students were selected for the experiment. In this, each student was given a programming question of finding the Least Common Multiple (LCM) of two numbers, that he/she had to code in a programming language of his/her own choice. They were given 45 minutes as maximum time to complete the code. Each student was given a computer with necessary software for programming. Out of 36 students, 24 students completed the given assignment on time correctly and the remaining 5 students could not complete the assignment in time and remaining 7 students completed the assignment but the results were wrong. The solutions given by the students were analysed based on the parameters like Start Time, End Time, Time complexity, Lines of Code, Correctness of Code. Most of the students selected “C” Language for completing the assignment.

Experiment 2 : Pair programming in Distributed environment

The experiment was conducted on 18 pairs formed with the 36 students considered for experiment 1. Each pair had one student from the Computer Science stream and one from the Biology Stream at their school level. The experiment was conducted in two different computer labs, situated at two different buildings within the same campus. Each pair was given a computer with all the required software including voice chat, desktop sharing, file transfer options. A head phone was provided for voice chatting.

Figure 3.1 Screen shot of the software used for voice chatting between the pairs

In this part each pair was given a programming question of finding whether a given string or number is a palindrome, that they had to code in a programming language of their own choice. They were given 45 minutes to complete the code. A special software was developed using Java for conducting the experiment. The software will prompt the pairs who are at different labs, to input the ip-address of their computer, so that a connectivity will be established between the pairs. The software has features for text chatting, audio chatting through a head phone. Files can be transferred through the computers between the pairs where connectivity is established. The desk top of the pairs can be viewed on their monitor for understanding the work done by the pairs. Short messages can also be send through this software.

The experiment was carried out in two computer labs where in each pair was given a computer with the required software which had the facilities for voice chatting, desktop sharing and file transfer options. The screen shot of the software used for voice and text chatting are shown in Figure 3.1 and Figure 3.2 respectively.

Figure 3.2 Screen shot of the software used for text chatting between the pairs

Out of 18 pairs, 16 pairs completed the given assignment correctly and one pair could not complete the assignment in time and another pair completed the assignment in time but shown wrong output.

The solutions given by the pairs were then again analysed based on the same set of parameters used for solo programming along with some additional parameters like level of interaction when distributed approach was used. Results are shown using various tables and graphs.

A questionnaire was given to the students to obtain a feedback about their experience in distributed pair programming practice. The questionnaire used to obtain the feedback is shown in Table 3.1. Table 3.2 is based on the feedback collected from students, who were involved in a programming lab assignment, which analyzes the effect of distributed pair programming on their performance. In this experiment students were asked to give a score ranging from 1 to 5, where 1 stands for strongly disagree, 2 for disagree, 3 for agreed to some extent, 4 for agree and 5 for strongly agree, on twenty metrics listed in the second column of the Table 3.2.

Table 3.1 Questionnaire for analyzing the quality of DPP

1. Strongly disagree 2. Disagree 3. Agree to some extent 4. Agree 5. Strongly agree

Sl.No.	Metrics	Statements	Score (1 to 5)
1	Confidence	Collaboration with my pair gives me more confidence in solving programming problems	
2	Confidence	Collaboration with my pair gives me more confidence in writing programs	
3	Navigator Role	Navigator role is very much important in guiding the driver to obtain the output	
4	Knowledge	I have gained more new knowledge by participating in the pair programming forum	
5	Learning process	The discussion in the forum is more focused towards the problem that need to be solved	
6	Learning experience	Collaborating with my peer in solving the given task is a new rewarding experience	
7	Peer	My team member gives concrete ideas	
8	Persistence	This collaborative learning should be expanded to other courses too	
9	Time	Less time is taken to obtain the solution using this approach	
10	No. of members	This collaboration in the forum will be more effective if it has more than two members	
11	Replacement	Pair collaboration with peer could replace the tutorial classes	
12	Satisfactory Level	The forum provided in tool to collaborate with the peer is enough and no other facilities is required	
13	Performance	Pair programming let people solve problems faster than traditional programming	
14	Learning time	Pair programming provide a mechanism for continuous and incremental learning	
15	Code optimization	Pair programming foster a better comprehension of the piece of code to be developed	
16	Cost / Resources consumption	Pair programming entail a greater overhead than traditional development	
17	Difficulty with pair	I find social difficulties with my companion	
18	Social Rules	Social rules should be observed when forming pairs	
19	Communication	Communication limitations modify significantly the quality of results during distributed pair programming	
20	Cooperation	Distributed Pair programming foster breaking down cooperative work	

The Table 3.2 shows the percentage of students agreeing on each score category for all the specified twenty metrics.

Table 3.2 Metrics for Data Analysis for DPP

Metric No.	Metrics	Strongly Disagree (%)	Disagree (%)	Agree to some extent (%)	Agree (%)	Strongly Agree (%)
1	Confidence increases while solving programming problems	0	0	25	33.33	**41.67**
2	Confidence increases while writing Programs	0	2.78	27.78	33.33	**36.11**
3	Navigator role is very important in guiding the driver role	0	8.33	27.78	**47.22**	16.67
4	Knowledge increases for pairs	2.78	11.11	27.78	**36.11**	22.22
5	Discussion is more focused towards the problem	0	5.55	16.67	**55.56**	22.22
6	Learning experience is improved	0	5.56	11.11	**50**	33.33
7	Team member gives good ideas	0	0	33.33	**41.67**	25
8	This methodology should be expanded to other subjects also.	0	0	16.67	33.33	**50**
9	Less time taken to obtain solution	2.77	2.78	27.78	30.56	**36.11**
10	If more than 2 members are there this exercise will be more effective	8.33	**33.33**	22.22	16.67	19.45
11	This methodology will replace tutorial classes	8.33	11.11	25	27.78	**27.78**
12	Satisfactory Level. Facilities provided are OK.	0	33.33	**47.22**	16.67	2.78
13	Problems are solved faster.	0	0	16.67	**61.11**	22.22
14	This method provides continuous and incremental learning	0	2.78	19.44	**41.67**	36.11
15	Code Optimization is possible	0	5.56	19.44	**41.67**	33.33
16	Cost or Resource Consumption is reduced	2.78	8.33	**41.67**	33.33	13.89
17	Difficulty with pair is found	**58.33**	30.56	8.33	2.78	0
18	Social Rules should be observed when forming pairs	22.22	25	**33.34**	11.11	8.33
19	Communication limitations affect quality of results	0	11.11	**75**	11.11	2.78
20	It builds up Cooperation among pairs	13.89	22.22	**41.67**	19.44	2.78

3.4 INFERENCE FROM METRICS ANALYSIS TO MEASURE THE EFFECTIVENESS OF DPP

It can be seen from the Table 3.2 that (from the entries which are marked bold) distributed pair programming gives students more confidence and increases the efficiency of the overall output of the work assigned to them. Data analysis for metric 1 and metric 2 from the table reveals that the confidence level increases in solving a problem in case of distributed pair programming, as there is a knowledge sharing between the pairs. In case of metric 4, 11.11% of students express that knowledge level does not increase in case of distributed pair programming and 36.11% of students agree that knowledge increases in case of distributed pair programming. This is due to the communication barriers between the pairs and lack of pair compatibility. In case of metric 10, majority of students feel that involvement of more than two people in this experiment, will result in confusion and difference of opinion. Metric 11 reveals the fact that, 25% of students feel that this methodology is not an replacement of tutorial classes, as in tutorial classes a well experienced faculty is involved in the teaching-learning process. Data analysis for metric 12 from the table reveals the fact that the satisfactory level for facilities provided for the distributed pair programming practice is slightly less than our expectation.

This shows that further analysis is required to improve the software tool used for distributed pair programming practice. From metric 17, one can see that, majority of students did not face any difficulty with his/her pair. Data Analysis for metric 19 reveals that the communication barrier affects the quality of results. So, further analysis is required for improving the communication procedure and the tool. Data Analysis for metric 20 reveals that the cooperation among pairs is build up to some extent only. So, further analysis is required for improving the cooperation among pairs either by modifying the software tool or process. This may be also due to communication defectiveness, lack of behavioral protocols and comparative experience of pairs.

The five pie charts shown from Figure 3.3 to Figure 3.7 give the pictorial representations of the percentage of students support for each five score category for the 20 metrics shown in the Table 3.2. From the pie chart shown in Figure 3.3 one can see that majority (Metric No.17) of the students do not agree with the concept that difficulty is experienced with the pair which is in support of distributed pair programming practice. Metric 20 reveals the fact that most the programmers do not agree that cooperation among the pairs is not build, because all the pairs selected are from first year of their programme and due to less amount of time in interacting with each other during the experiment, they feel that much cooperation cannot be built up. Metric 18 reveals that many students strongly disagree with the concept that social rules should be observed when forming pairs. This shows that the students do not like strict rules and regulations during pair programming.

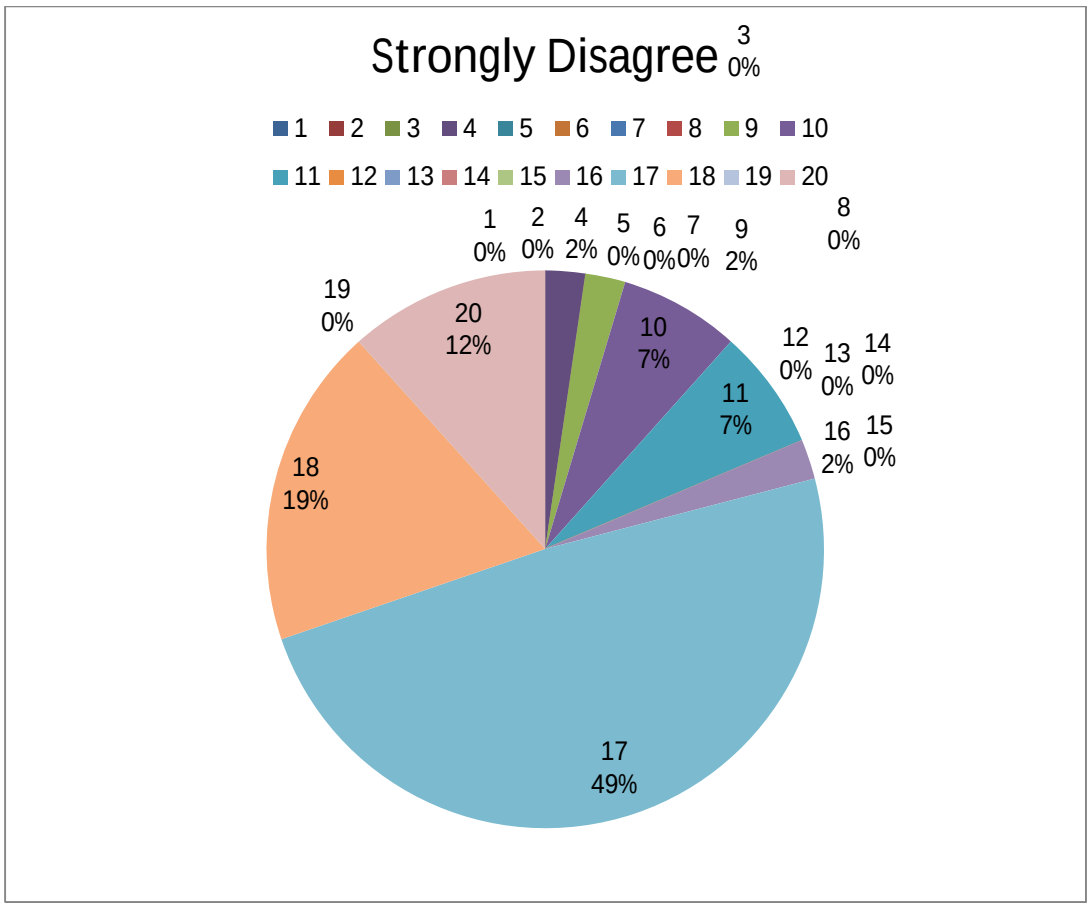


Figure 3.3 Data analysis of 20 metrics for the feedback "STRONGLY DISAGREE" from the students

12% of the students feel that cooperation among the pairs is not built up, because all the pairs are from their first year of their M.Sc (Software Engineering) programme and without knowing each other to a large extent, they may feel that cooperation may not be there, just because of the interaction during pair programming.

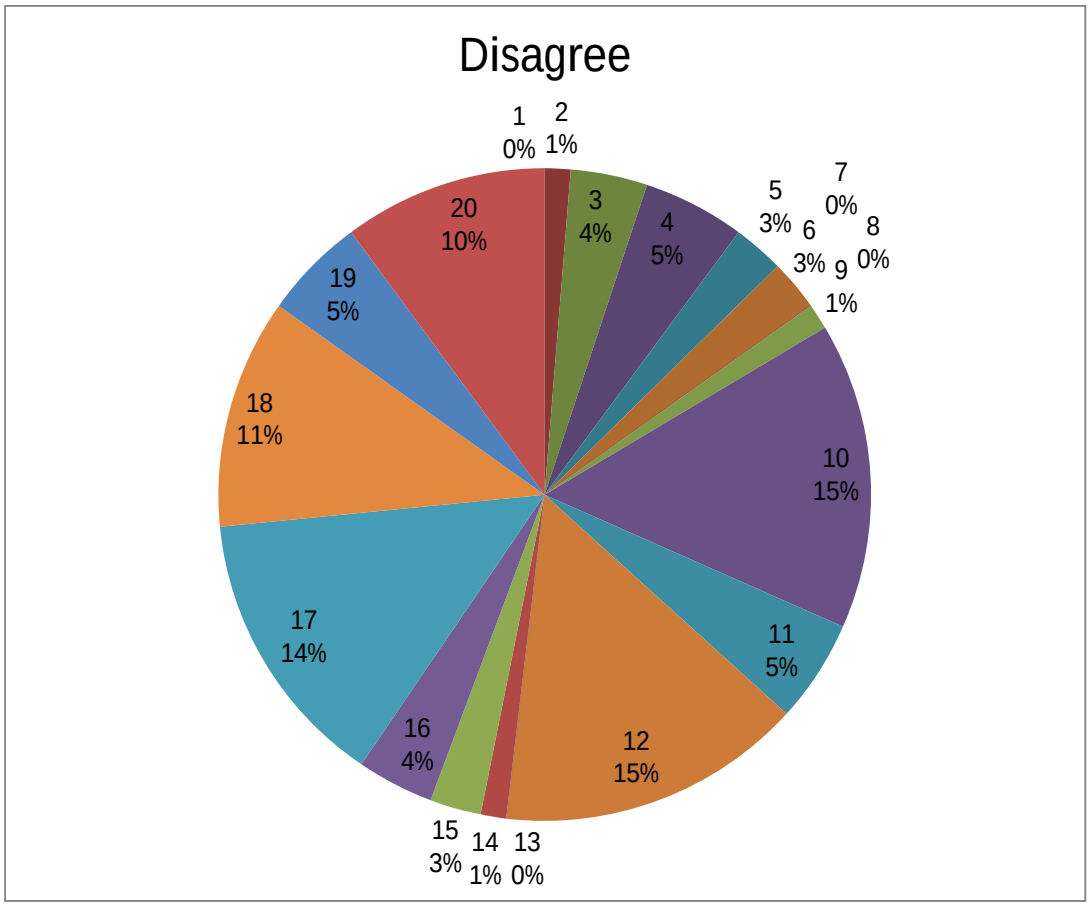


Figure 3.4 Data analysis of 20 metrics for the feedback "DISAGREE" from the students

The pie chart shown in Figure 3.4 shows that some of the students (Metric No.12) feel that distributed pair programming do not provide a satisfaction and facilities provided for the experiment are not sufficient. This needs further investigation and this may be due to the need for additional procedures and facilities for incremental learning. Metric 17 reveals that

many students did not have any difficulty with his /her pair during pair programming. This indicates that the pairs are compatible with each other.

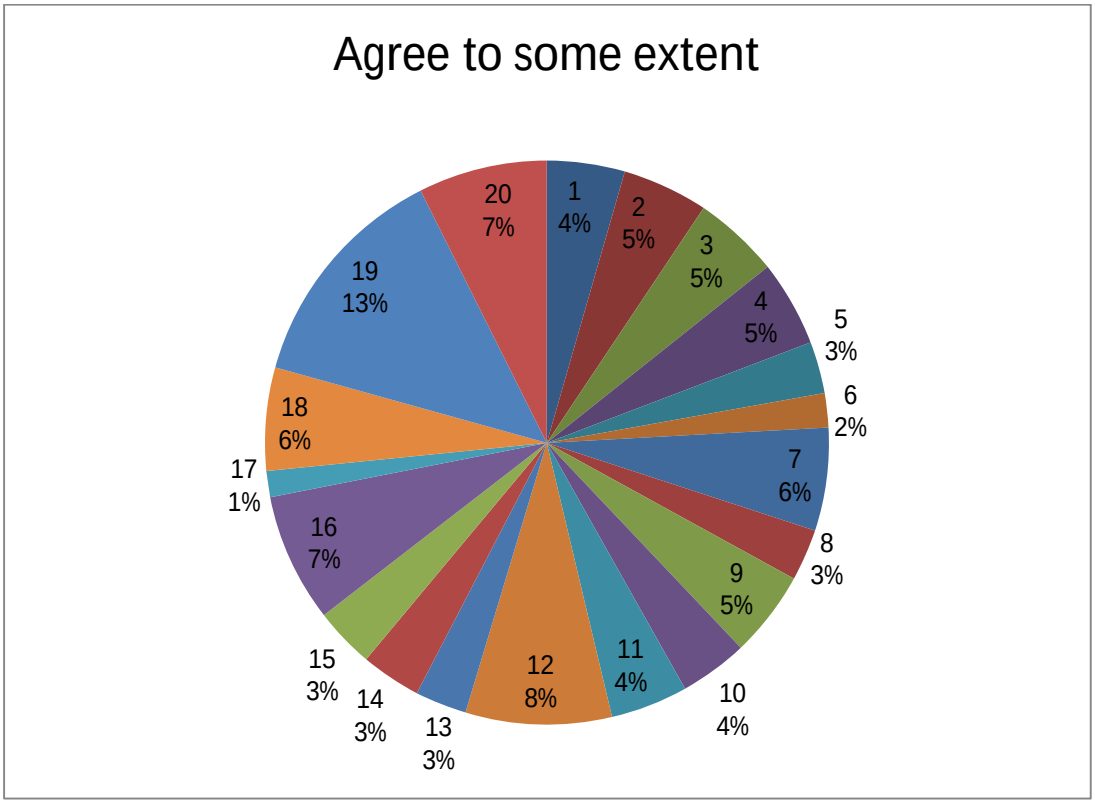


Figure 3.5 Data analysis of 20 metrics for the feedback "AGREE TO SOME EXTENT" from the students

The pie chart shown in Figure 3.5, reveals the fact that communications limitations affect quality of results to a certain extent. This shows that the communication tool and procedure needs further improvement. (Metric No.19). This reveals the fact that better software tools are necessary for communication between the pairs. 7% of the students feel that the cost or resource consumption is reduced. This is due to the fact that, communication takes place through audio chatting or by sending the message to the pair. File transfer feature is also enabled in the communication software. All these features reduce the cost consumption. Knowledge transfer also helps them to solve the problem without taking much time. This is another useful mechanism where the usage of resources are not wasted.

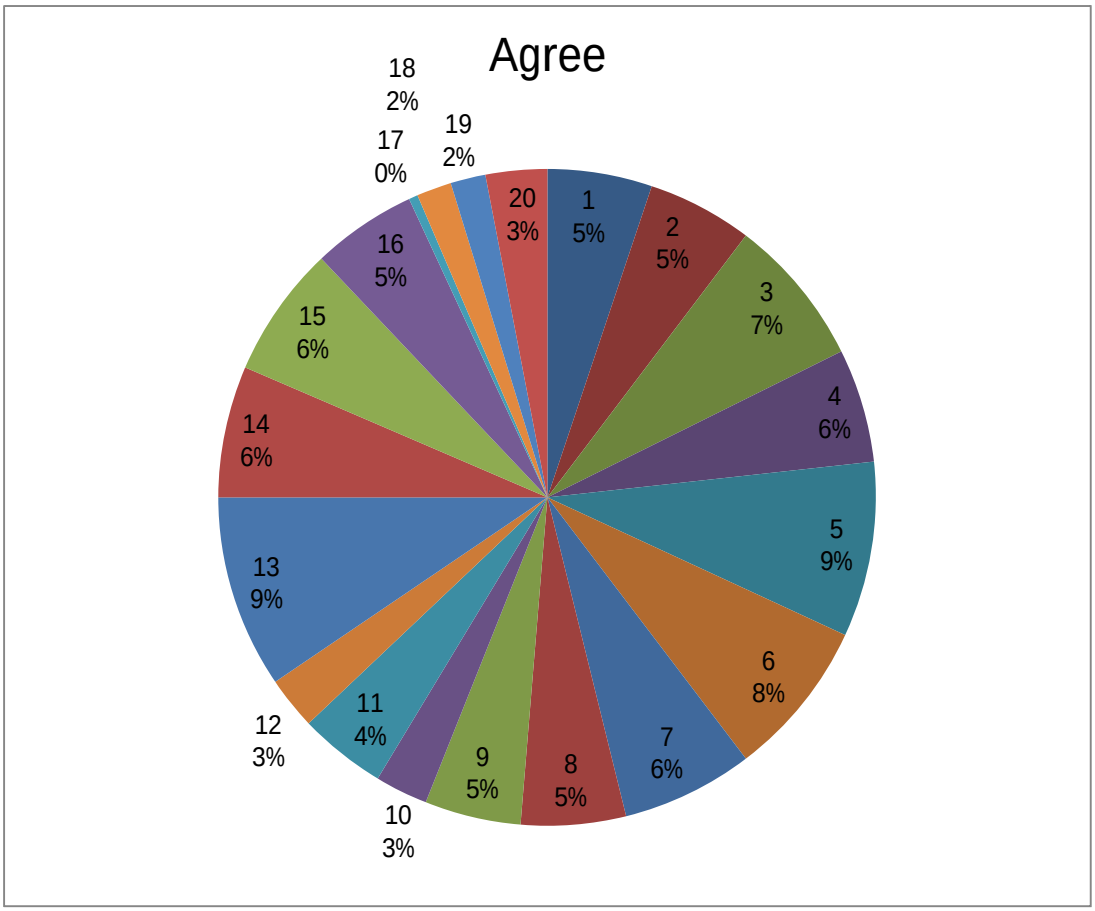


Figure 3.6 Data Analysis of 20 metrics for feedback "AGREE" from the students

The pie chart shown in Figure 3.6, reveals that majority of the students agree that (Metric No.13) problems are solved faster during the pair programming practice where, time is saved when compared with solo programming practice. Metric No. 5, indicates that discussion is more focused towards the problem during pair programming. Since only two people are involved the discussion is more focused on the problem and it may not divert to other issues. Programmers also agree that the team member gives good ideas (Metric No. 7) and learning experience is improved. (Metric No.6)

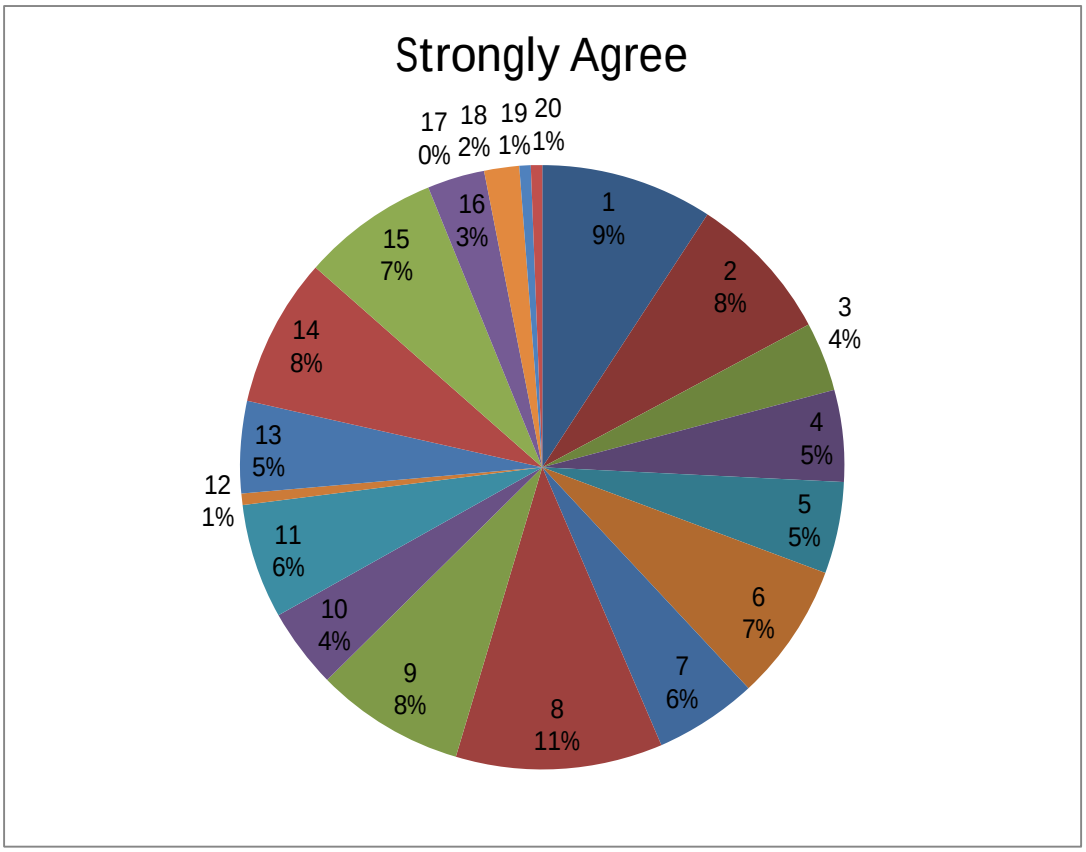


Figure 3.7 Data analysis of 20 metrics for the feedback "STRONGLY AGREE" from the students

The pie chart shown in Figure 3.7, reveals that most of the students strongly agree that the distributed pair programming practice should be expanded to other subjects also, which in support of distributed pair programming practice(Metric No.8). The students feel that this methodology will help in solving the problems without much difficulty in case of complex courses as there is knowledge sharing between the pairs and innovative ideas may emerge during the communication process between the pairs. Metric 1 and 2 indicates that confidence level increases during pair programming which is strongly agreed by many students. Metric 9 indicates that with in a short period of time, students were able to solve the problems and they could obtain a solution.

3.5 INFERENCE FROM LINES OF CODE ANALYSIS TO MEASURE THE EFFECTIVENESS OF DPP

Based on number of lines of code and time taken by them to code, graphs shown in Figure 3.8 and Figure 3.9 were plotted. Graphs shown in Figure 3.8 plots the average lines of code by all students against the time interval when the output of the code is correct.

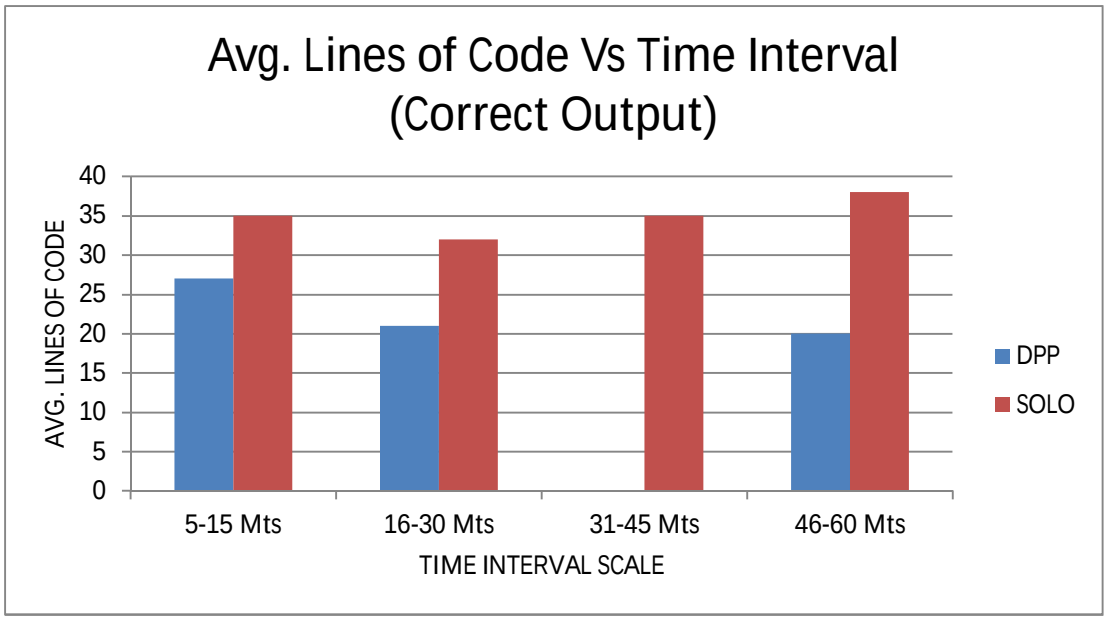


Figure 3.8 Lines of Code analysis for correct outputs

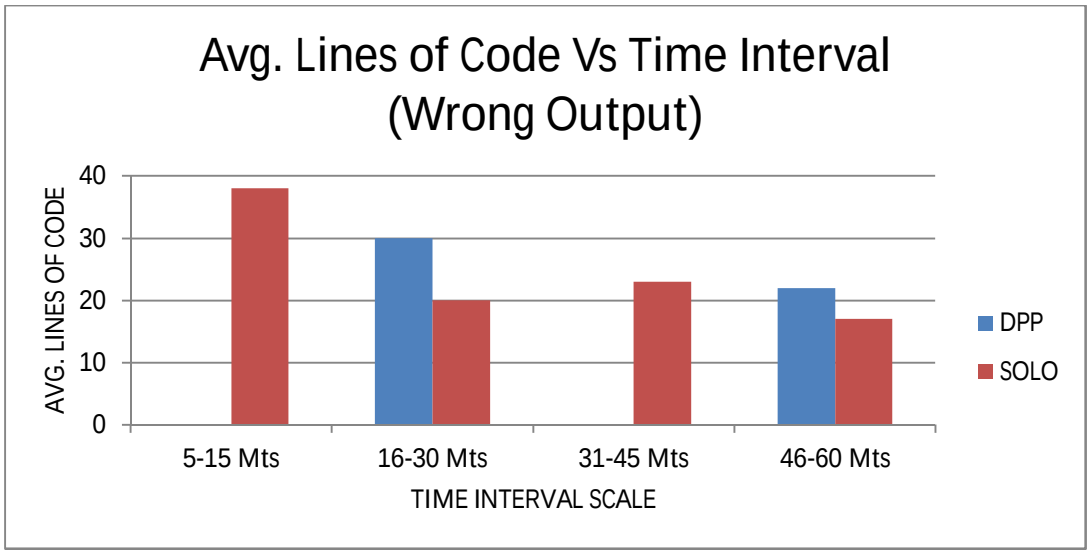


Figure 3.9 Lines of Code analysis for wrong outputs

The four time intervals are 5-15, 16-30, 31-45 and 46-60. Time Intervals are given in minutes. Series1 represents distributed pair programming and Series 2 represents solo programming. In Figure 3.8 we can observe that in distributed pair programming, the average lines of code is always less when compared to the solo programming in all the four time intervals in case where the output is correct. It is observed from Figure 3.8, that there was no student who completed the code with correct output in the case of distributed pair programming experiment in the time interval ranging between 31 to 45 minutes. Figure 3.9 plots the average lines of code by all students against the time interval when the output of the code is wrong. It is observed that in all the four time intervals solo programmers exist with wrong outputs.

3.6 INFERENCE FROM DEFECT ANALYSIS TO MEASURE THE EFFECTIVENESS OF DPP

Figure 3.10 shows the graph which plots the number of students who got wrong outputs in the four time intervals for both distributed pair programming and solo programming. Series 1 represents distributed pair programming and Series 2 represents solo programming. It is observed from the Figure 3.10 that there were no students in the case of distributed pair programming who got wrong outputs in the time interval ranging between 5 to 15 minutes and 31 to 45 minutes.

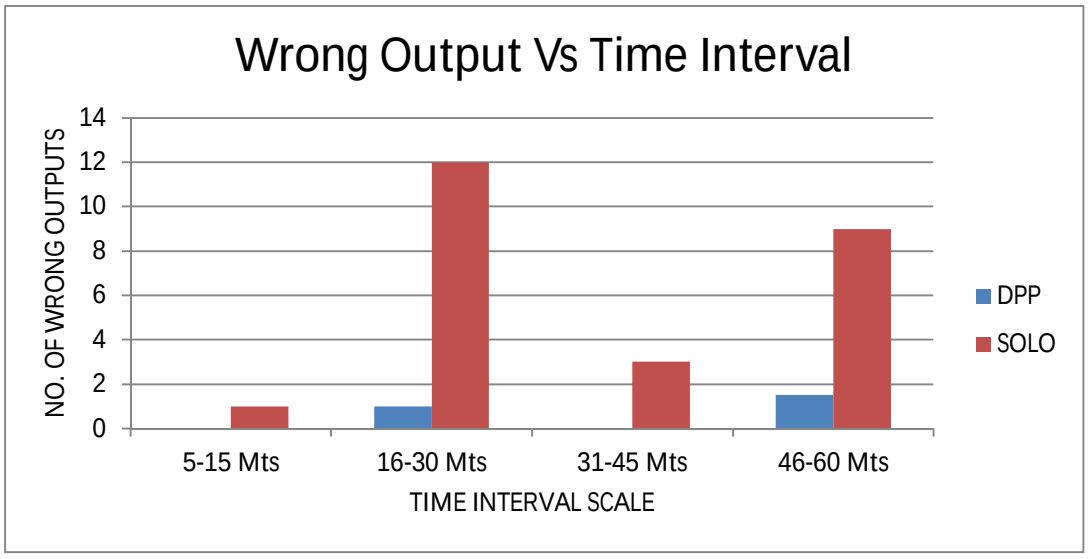


Figure 3.10 Time Analysis for wrong outputs

Figure 3.11 shows the graph which plots the number of students who got correct outputs in the four time intervals for both distributed pair programming and solo programming. It is observed from the Figure 3.11 that there were no students in case of solo programming, who got the correct outputs in the time intervals ranging between 5 to 15 minutes, 16 to 30 minutes and 31 to 45 minutes.

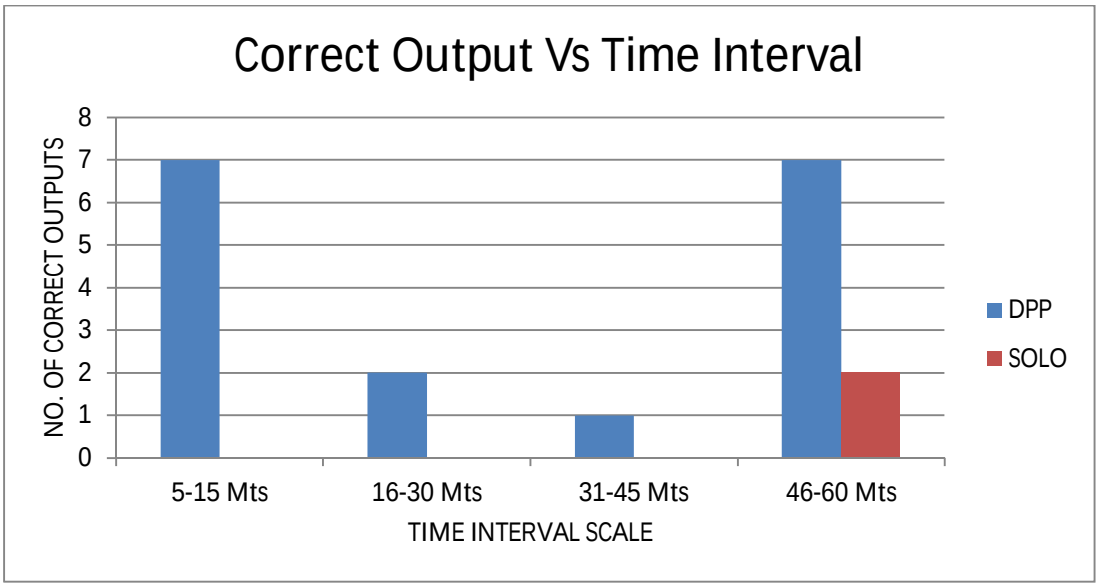


Figure 3.11 Time Analysis for correct outputs

This is due to the fact that since the students are in the first year of their Five Year Integrated M.Sc (Software Engineering), they took some more time for completing the experiment with correct output. One can see that wrong outputs are maximum in case of solo programming and minimum in case of distributed pair programming. It is also observed that maximum number of correct outputs were obtained in case of distributed pair programming and minimum number of correct outputs were obtained in the case of solo programming.

3.7 CONCLUSION

Experiments have been conducted to evaluate the performance of students who engaged in distributed pair programming during laboratory courses with those who worked solo during the laboratory sessions. Even though the laboratory sessions did not add directly to the final grade, the outcome of being involved in a distributed pair programming experience appear to have enhanced the quality of assignment work. Furthermore, the majority of students enjoyed the practice and would like to have distributed pair programming in courses. The results provide the support for use of distributed pair programming practices in the software engineering curriculum. Future work will verify the results by repeating the experiments again for the forthcoming laboratory courses and for finding out more about the students who did not enjoy the distributed pair programming practices.

CHAPTER 4

WEIGHTED GRAPH MATCHING APPROACH FOR PAIR COMPATIBILITY IN PAIR PROGRAMMING

Pair compatibility plays a significant role in the performance of a pair in the given lab assignment. Weighted graph matching technique is proposed in forming pairs with high degree of pair compatibility.

4.1 INTRODUCTION

Pair programming is a popular practice under Extreme Programming. It is increasingly followed in IT industries for effective software development and in many educational institutions for laboratory assignments. Traditionally, pairs are formed based on individual preferences or administrative authority's decision to support organization requirements as there are no standard procedures for forming pairs. A pair's performance will be highly productive if they are more compatible with each other. In this chapter, a novel method is proposed to form student pairs for programming laboratory based on weighted graph matching technique incorporating necessary psychological factors for compatibility between pairs. The experimental results demonstrate that the proposed method yields better performance of the pairs.

4.2 SIGNIFICANCE OF PAIR MATCHING

Extreme programming (XP) is a kind of agile software development methodology that stresses on achieving customer satisfaction through team work and focuses on bringing high productivity. Pair

programming is one among the various principles of XP and it has been followed in many software industries and universities especially for tasks related with software development.

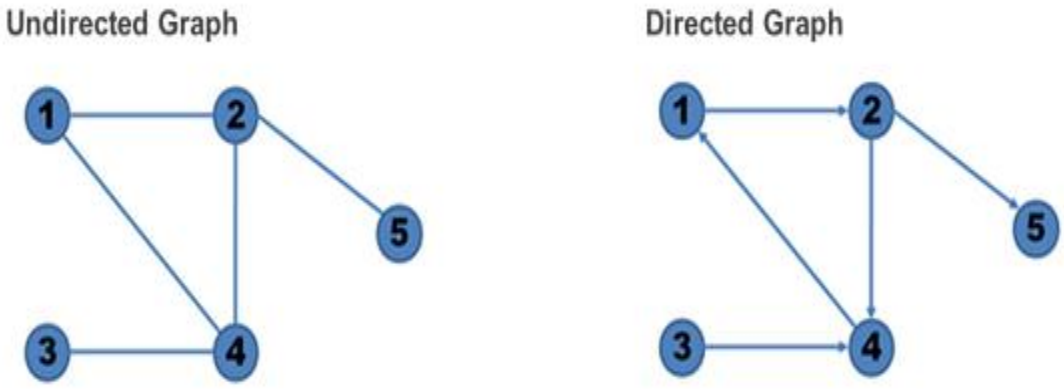
Despite the benefits of pair programming, there are also some negative views about it. (Tessem 2003), showed that some students found the experience so irritating, inefficient and exhausting. Very similar results were found by (Gittins & Hope 2001). In their study, participants described the experience with pair programming as demanding and sometimes frustrating. Moreover, (VanDeGrift 2004) showed that the students complained about working among people with different personalities and skill levels. Also, Lucas Layman's study (Layman 2006) on the effects of collaborative work on students cited non participatory partners and difficulties in scheduling discussion times outside the classroom as major reasons for students disliking pair programming. In spite of these negative outlooks, new methods are appearing with comparable performances and greatly successful outcomes giving credence to the idea of pair programming. Generally, it is always preferable to have a companion who could be supportive in achieving the target, instead of performing the assigned task by working all alone. Programming is not an exception to this conviction.

A challenging task in pair programming is to anticipate and measure the potential compatibility between individuals thereby maximizing the productivity. As there is no principle to evaluate partner compatibility, earlier studies on pair compatibility suggested to form pairs based on various personality factors (Katira et al. 2004). The failure rate related with pair programming experiments can be reduced to a greater extent by formulating an appropriate measure of compatibility.

4.3 GRAPH MATCHING APPROACH

In this section the graph theory related definitions and terminologies which help in modeling the pair programming problem using graph matching concepts are discussed.

A graph G = (V,E) is a mathematical structure consisting of two sets V and E. The elements of V are called *vertices* or *nodes* and the elements of E are called *edges.*

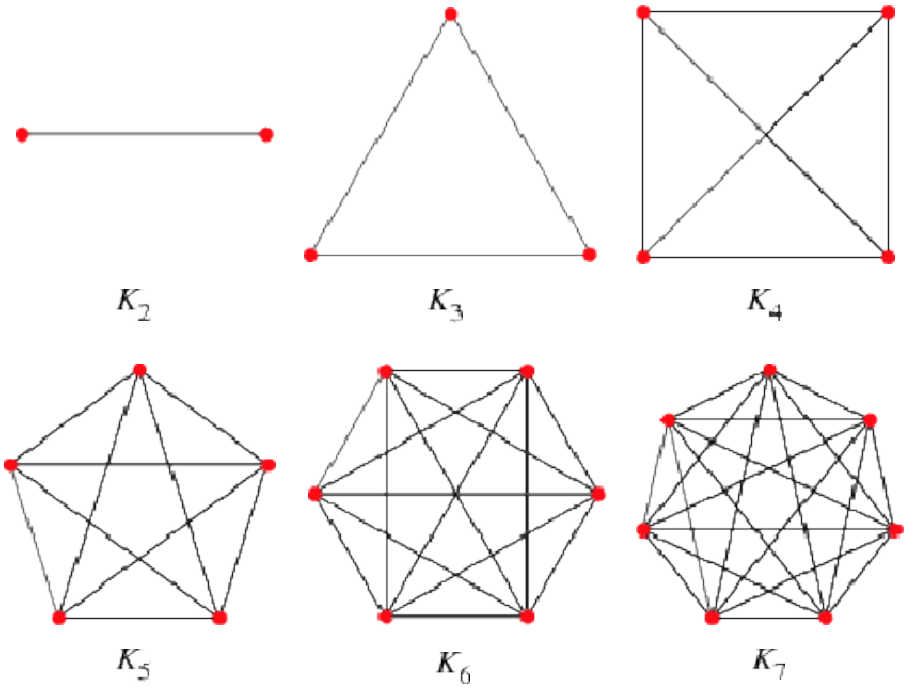


(a) **(b)**

(Source : http://www.alberton.info)

Figure 4.1 Undirected graph and Directed graph

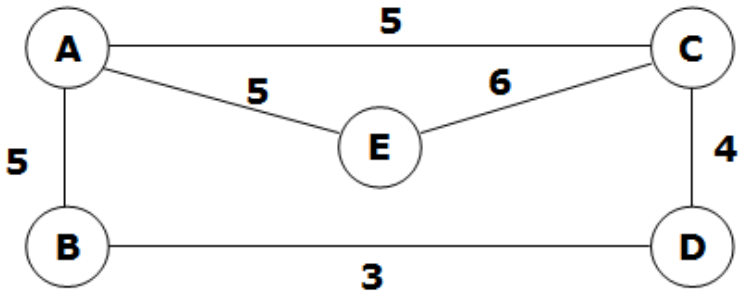
An *undirected graph* G is defined by a set V(G) of elements called vertices, a set E(G) of elements called edges, and a relation of incidence, which associates with each edge an unordered pair of vertices called its end vertices. An example is shown in Figure 4.1 (a) for an undirected graph and Figure 4.1 (b) for a directed graph. There are 5 vertices or nodes in the graph, numbered from 1 to 5.



(Source : http://mathworld.wolfram.com/CompleteGraph.html)

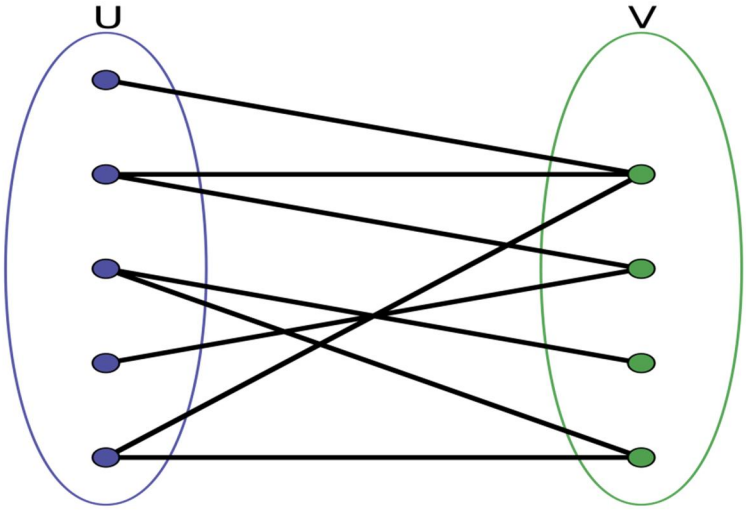
Figure 4.2 Complete graphs

A *Complete graph* is a simple graph such that every pair of vertices is joined by an edge as shown in Figure 4.2. Any complete graph on n vertices is denoted by K_n. A *complete graph* K_n on *n* vertices is one in which an edge is drawn from each vertex to every other vertex in the graph, resulting in a total of nC_2 edges.



(Source: http://www.multiwingspan.co.uk)

Figure 4.3 Weighted graph

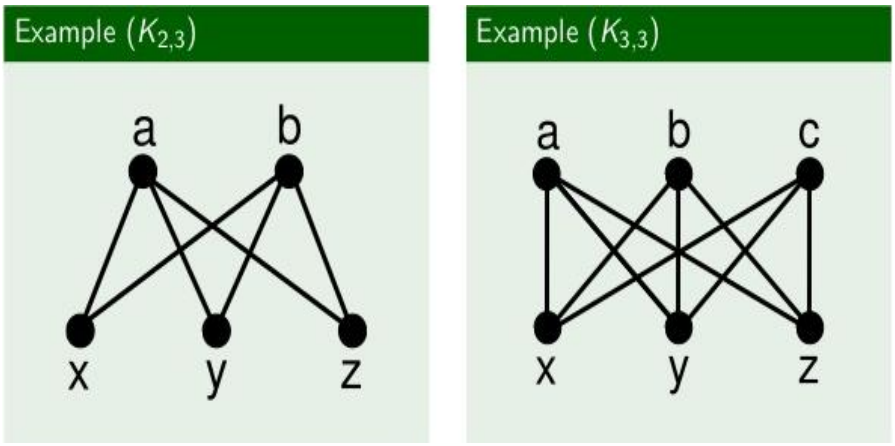


(Source: https://en.wikipedia.org/wiki/Bipartite_graph)

Figure 4.4 Bipartite graph

A weight is a numerical value, assigned as a label to a vertex or edge of a graph. A weighted graph is a graph whose vertices or edges have been assigned weights; more specifically, a vertex-weighted graph has weights on its vertices and an edge-weighted graph has weights on its edges. The weight of a subgraph is the sum of the weights of the vertices or edges within that subgraph. If a real value is assigned to every edge of G, then G is called a *weighted graph* as shown in Figure 4.3.

Given a **weighted graph**, and a designated node S, we would like to find a path of least total weight from S to each of the other vertices in the graph. The total weight of a path is the sum of the weights of its edges. A *bipartite graph* is a graph whose vertex set can be partitioned into two disjoint sets U and V such that every edge connects a vertex in U to a vertex in V. An example for bipartite graph is shown in Figure 4.4.



(a) **(b)**

(Source: http://www.slideshare.net/uyar/graphs-7802324)

Figure 4.5 Complete bipartite graphs

A *complete bipartite graph*, denoted by $K_{m,n}$ is a bipartite graph where the two partitions X and Y are of sizes m and n respectively and every vertex in X is connected to every vertex in Y. In Figure 4.5 the examples are shown for complete bipartite graphs denoted by $K_{2,3}$ and $K_{3,3}$.

A *matching* M in a simple graph is a collection of mutually non adjacent edges. The vertices incident to the edges of a matching M are saturated by M; the others are unsaturated. A maximal matching in a graph is a matching that cannot be enlarged by adding an edge. A maximum matching is a matching of maximum number of edges among all matching's in the graph. Every maximum matching is maximal, but not the converse. A perfect matching is a matching in which all the vertices of G are saturated. *Maximum Weighted Matching* (MWM) of a weighted graph is a matching in which the sum of the weights of the edges is maximum. Mathematically it can be represented as,

$$MWM = maximize \sum_{e \in M} w(e)$$

where $w(e)$ indicates the weight of the edge e.

All possible maximum weighted matching's of a 4-vertex complete weighted graph is shown in Figure 4.6.

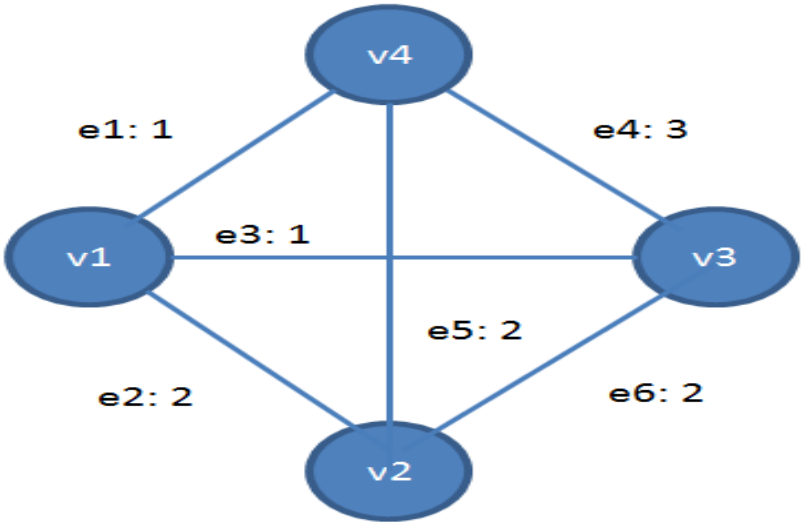


Figure 4.6 Matching of a weighted complete graph

In this graph, the match M_1 is the compatibility between edge e1 vertices v1 and v4 and compatibility between the edge e6 vertices v2 and v3.

Match M_1 = sum of edge weights of {e1, e6} = 3

Match M_2 = sum of edge weights of {e2, e4} = 5

Match M_3 = sum of edge weights of {e3, e5} = 3

Here the non adjacent edges are considered for matching according to the Edmonds' blossom algorithm. Match M_2 is the compatibility between edge e2, vertices v1 and v2 and the compatibility between edge e4 vertices v3 and v4. Match M_3 is the compatibility between the edge e3 vertices v1 and v3 and the compatibility between edge e5 vertices v2 and v4. Since the weight of match M_2 is the maximum when compared with the weights of

other matches M_1 and M_3, we consider M_2 for pair compatibility. M_2 is considered as the maximum weighted matching.

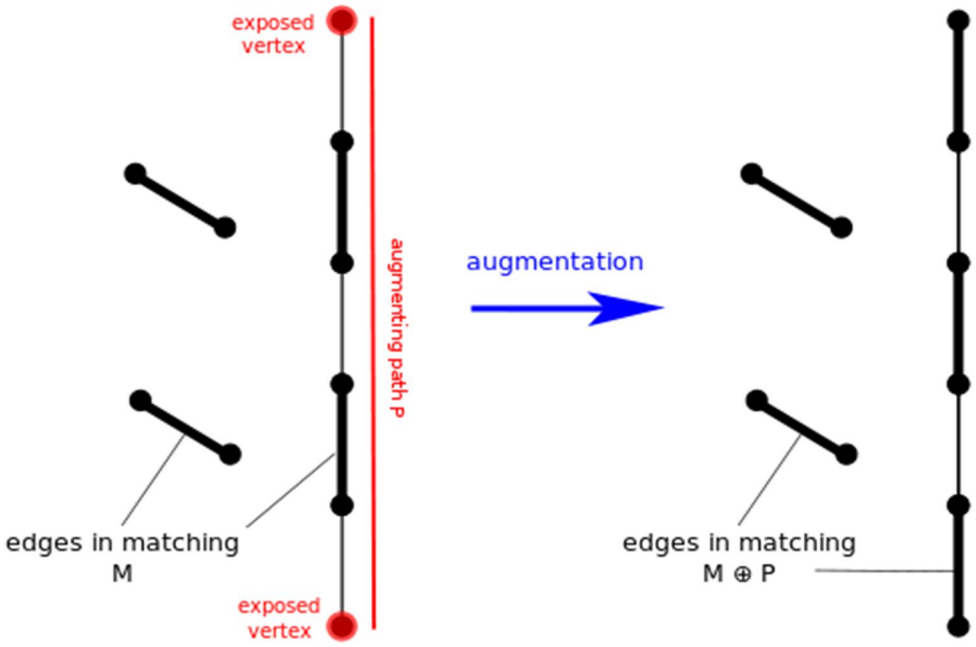
The concept of graph matching is used in many industrial applications. In industrial planning, a wide variety of assignments are routinely made. For instance, the assignment of individual workers to tasks, jobs to processors, etc., can be modeled using graph matching. In some scenarios like man-machine assignment, the industry wishes to consider the experience of the person in handling the machine. Weighted bipartite graphs can be used to represent this situation. Various algorithms have already been proposed for finding matching's in general graphs. Variations of Edmonds' Blossom and Hungarian algorithms are generally used for finding maximum weighted matching in complete graphs and complete bipartite graphs respectively. For more concepts on graph theory (West 2011) is a good reference.

4.3.1 Edmonds' Blossom Algorithm

The blossom algorithm is an algorithm in graph theory for constructing maximum matchings on graphs. The algorithm was developed by Jack Edmonds in 1961 and published in 1965. Given a general graph $G = (V, E)$, the algorithm finds a matching M such that each vertex in V is incident with at most one edge in M and $|M|$ is maximized. The matching is constructed by iteratively improving an initial empty matching along augmenting paths in the graph. Unlike bipartite matching, the key new idea is that an odd-length cycle in the graph (blossom) is contracted to a single vertex, with the search continuing iteratively in the contracted graph.

4.3.1.1 Augmenting paths

Given $G = (V, E)$ and a matching M of G, a vertex v is **exposed** if no edge of M is incident with v. A path in G is an **alternating path**, if its edges are alternately not in M and in M (or in M and not in M). An **augmenting path** P is an alternating path that starts and ends at two distinct exposed vertices. A **matching augmentation** along an augmenting path P is the operation of replacing M with a new matching $M_1 = M \oplus P = (M \setminus P) \cup (P \setminus M)$.



(Source : http://www.contrib.andrew.cmu.edu)

Figure 4.7 Match augmentation in Edmonds' Blossom algorithm

It may be proven that a matching M is maximum if and only if there is no M-augmenting path in G. Hence, either a matching is maximum, or it can be augmented. Thus, starting from an initial matching, we can compute a maximum matching by augmenting the current matching with augmenting

paths as long as we can find them, and return whenever no augmenting paths are left. We can formalize the algorithm as follows:

INPUT : Graph *G*, initial matching *M* on *G*

OUTPUT : maximum matching *M** on *G*

A1 **function** *find_maximum_matching*(*G*, *M*) : *M**

A2 *P* ← *find_augmenting_path*(*G*, *M*)

A3 **if** *P* is non-empty **then**

A4 **return** *find_maximum_matching*(*G*, augment *M* along *P*)

A5 **else**

A6 **return** M

A7 **end if**

A8 **end function**

4.4 RELATED WORK

A good deal of research has already been carried out on pair programming from academic as well as industrial perspective. The recent industrial case study (Bella et al. 2005) analyzed that the relationship between pair programming and defect rate under various scenarios for a team of an Italian company shows that pair programming appears to provide a perceivable but small effect on defect reduction. However, their results also

indicate that the introduction of new defects tend to decrease when pair programming is practiced.

Salleh et al. (Salleh et al. 2011) presented a systematic literature review of empirical studies that investigated factors affecting the effectiveness of pair programming for Computer Science/Software Engineering students. Their literature review listed 14 compatibility factors like personality types, actual and perceived skill levels, gender, communication skills, learning style, etc., that affect pair compatibility and/or pair programming's effectiveness as a pedagogical tool. In their study, skill level was identified to be the most important factor amongst all and their results showed that a pair works well when both students have similar abilities and motivation to succeed in a course.

On the contrary to other studies focusing on the effects of pair programming on software quality and development time, the study by Sillitti et al. (Sillitti et al. 2012) focuses on the effects that pair programming has on developers' attention and productivity. According to their study, people working in pairs concentrate more on productive activities and they engage themselves in significantly longer and uninterrupted working sessions; they focus more on the assigned tasks and thereby the need for retrieving information from sources other than the partner for example, using web is also reduced.

Chaparro et al. (Chaparro et al. 2005) employed various data gathering techniques and analysed the reason for ineffectiveness of pair programming. In their study, students' skill levels and the programming tasks are identified to play a major role in the effectiveness of pair programming process. They also suggested that students should be matched with a partner who has similar skill level and a novice student should always be paired with

a partner with higher skill level. Thus we find that many of the studies on pair programming insist on skill level and compatibility for effective results.

4.5 METHODOLOGY

Owing to the ever increasing needs for pair programming, industries and academics use pair programming more frequently, but they often find it hard to understand the underlying principles of forming pairs. The success of pair programming is determined through the pairs' success which depends on how compatible they are with each other and how effectively they can communicate and understand each other's thoughts. Incompatible pairs have less understanding and improper communication between them. This leads to reduced performance, demotivation / frustration and disengagement from work. The consequences of having incompatible pairs also include delay in meeting deadlines, higher development time even for a smaller module and poor quality work. So pair compatibility is indeed essential for success of pair programming. A pair is said to be effective if they can produce better performance which arises out of their compatibility. Generally in the case of social and student networks, the most efficient mechanism is to adopt a few psychological factors of human minds and a measure of skill level.

Here a novel method is proposed to measure the compatibility between individuals taking into account their skill levels too. Let G be a weighted graph formed by considering the students as vertices and the edge weights represent the measured compatibility between the corresponding pairs of individuals. The problem is to pair the students such that the overall compatibility of pairs is maximized. So according to graph theory, the problem is equivalent to finding maximum weighted matching in G. The concepts of pair programming and graph matching restricts that a student can have only one partner. Also since no student should be left out without a

partner, a perfect matching is required and to find one, the number of students participating is taken to be even in number in all the experiments.

The students were prepared for pair programming assignment by conducting a briefing session on pair programming and they were instructed that they will be doing their work in pairs on a working day. Since, all the students have basic computer skills, we presume that there are no scheduling difficulties as they are available during the working time. With the assumptions that there is no gender discrimination; they were all interested in pair programming and participants did not quit the work in between, all the experiments were conducted as planned.

4.6 EXPERIMENTS AND RESULTS

This work is a case study which is conducted to explore the effectiveness of pairs using graph matching for compatible pairing of students. The success of a pair is determined by the accurate output of the program execution within the allotted time and the usage of effective coding standards. Three experiments were conducted on a same batch of first year students from the five year integrated M.Sc Theoretical Computer Science. 26 students volunteered to participate which led to 13 pairs working on a given coding problem in Data Structures using C Programming language for each experiment. The questions were given to them on all the three experiments and were of equal complexity levels.

All the experiments were allotted a maximum time of 140 minutes. The first experiment comprised of pairs on their own choice. The pairs for the second and third experiments were formed using graph matching method. In the second experiment we have tested the students for compatibility without considering their skill levels, whereas the skill levels of pairs were also taken into account for the third experiment.

4.6.1 Self Chosen Pairs

In the first experiment, the students were asked to pick partners of their own choice with no constraints. Therefore students preferred to work with their own friends. To evaluate the performance of pairs, Lines of Code and time taken by each pair were analyzed and compared. Though lines of code and time consumed do not have a direct impact on performance/outcome but, they are in general important factors for determining productivity. The result of the first experiment pairs with respect to these aspects is shown in Figure 4.8 and Figure 4.9. In Figure 4.8 the pairs are sorted according to lines of code is shown. In Figure 4.9 the time taken by the corresponding pairs in the first experiment is shown.

Table 4.1 Lines of code and Time taken for Experiment 1

Pa Pairs	Lines of Code	Time Taken (In minutes)
Pair 1	70	140
Pair 2	75	140
Pair 3	81	140
Pair 4	85	140
Pair 5	110	130
Pair 6	113	130
Pair 7	135	125
Pair 8	135	130
Pair 9	140	120
Pair 10	142	130
Pair 11	143	140
Pair 12	160	120
Pair 13	180	110

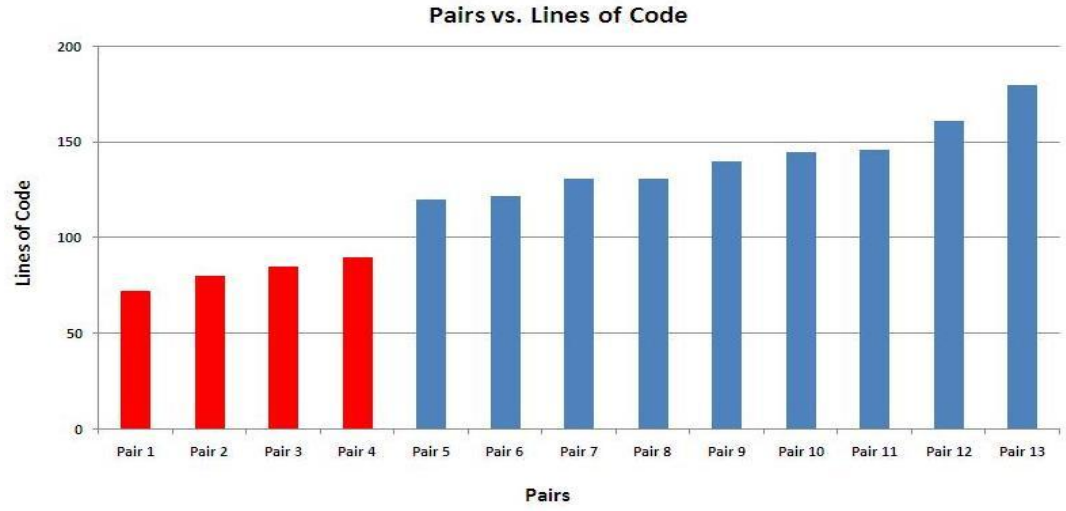


Figure 4.8 Lines of code analysis with respect to experiment 1

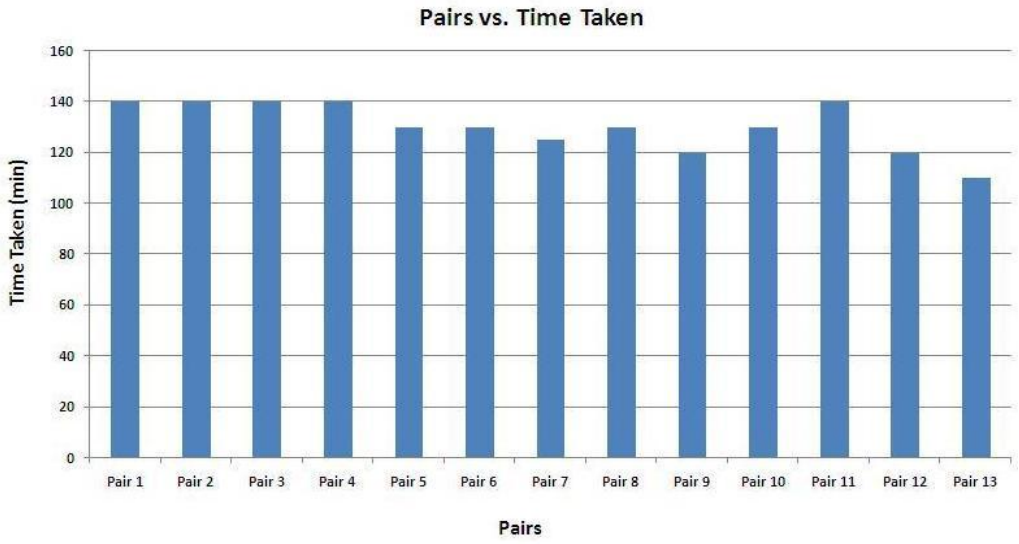


Figure 4.9 Time Analysis for the pairs with respect to experiment 1

In the experiment 1 four pairs namely, pair1, pair 2, pair 3 and pair 4 were unsuccessful as shown in Figure 4.8, because they were unable to complete the programming assignment successfully within the allotted time of 140 minutes. Figure 4.9 depicts the time taken by both the successful and unsuccessful pairs. We can also infer that the time taken by unsuccessful pairs

is more when compared to that of successful pairs; however it need not be true in general. Though unsuccessful pairs experienced some other benefits with their partners, these pairs yielded ineffective outcomes as both partners were found to be less competent. Table 4.1 shows the lines of code and time taken for experiment 1.

4.6.2 Pairs chosen using weighted graph matching without considering skill level

Here, it is assumed that individuals participating in pair programming experiment are of same experience levels and they have similar skill levels as peer groups are only considered here. To obtain correct information about each individual's personal characteristics which is necessary to assess the compatibility between individuals, every participant was given a questionnaire prior to conducting the experiment 1. The questionnaire consists of multiple choice questions to test the individual's skills, behaviors including working nature, decision making skills and temperament. Generally, any number of questions can be asked with n different options. Conventionally four options are preferred. A total of 15 questions were given with each question having 4 different options. The answers to each question were fixed by assigning credits to every option to the answer. The best option was given a credit 4, next best was given 3 and then 2 and finally least option was given a credit 1. With a belief that students assess themselves properly, every participant was assigned a score based on his/her answers. The scores were calculated considering only the answers to the questionnaire and no other additional factors were included. The student's answer to every question was cumulated to arrive at his/her final score. PS_{ij} indicates the predefined value to the option j chosen as answer by the student for the question i.

Then his/her absolute score QS_k is computed using the formula as shown in Equation (4.1).

$$\text{Score } QS_k = \frac{\text{sum of score to each answer}}{\text{total credits}} = \frac{\sum_{i,j} PS_{ij}}{\sum_{i=1}^{4} i} \tag{4.1}$$

where "k" is the serial number of the student.

From the absolute score, relative score S_k to each individual is then calculated using percentiles, to evaluate the relative standing of a student amongst all students. With relative scoring, the batch performance was improvised as a whole rather than each individual's performance.

Having computed the scores of every student "k", the student network was constructed as a weighted complete graph(K_{26}). The weight of vertex k was assigned to be the score S_k of kth student. With each vertex having relative score, the edge weight was computed for every edge by taking the average of its vertex weights. Suppose an edge e is incident on vertices v_1 and v_2, its weight w_e is shown in Equation (4.2).

$$w_e = \frac{(w_{v1} + w_{v2})}{2} \tag{4.2}$$

where w_{vi} indicates the weight (score) of the vertex(student) v_i .

These edge weights measure the level of compatibility between the corresponding individuals, which is a critical element for deciding the effectiveness of pairs. The graph with these calculated edge weights is then fed to Edmond's Blossom graph matching algorithm to find a maximum weighted matching. The pairs which were formed as a result of the algorithm are considered for the experiment 2.

Table 4.2 Lines of Code and Time taken for Experiment 2

Pa Pairs	Lines of Code	Time Taken (In minutes)
Pair 1	60	120
Pair 2	110	125
Pair 3	112	120
Pair 4	113	135
Pair 5	120	125
Pair 6	125	118
Pair 7	128	125
Pair 8	130	130
Pair 9	135	120
Pair 10	139	130
Pair 11	142	130
Pair 12	145	120
Pair 13	149	117

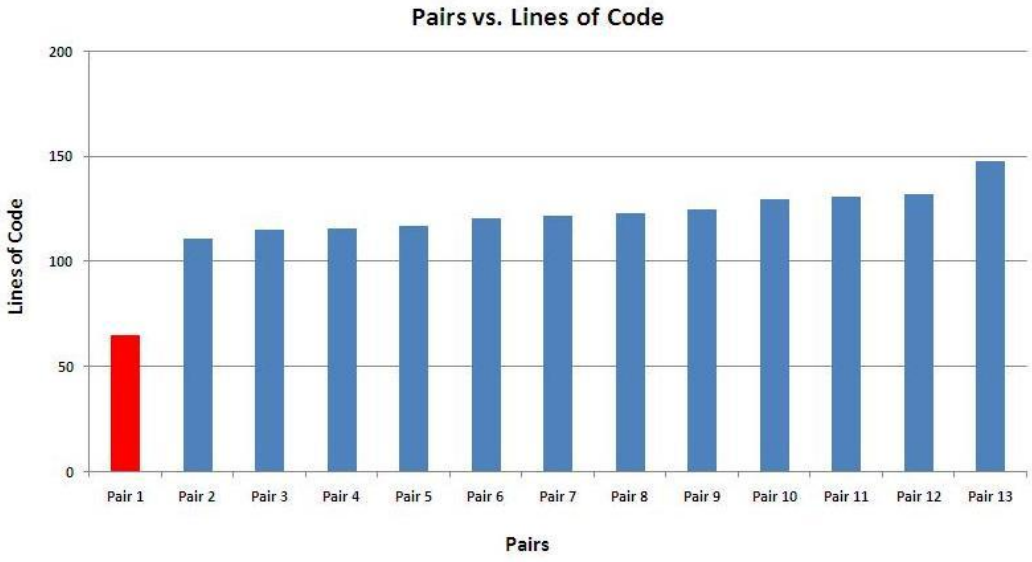


Figure 4.10 Lines of Code analysis with respect to experiment 2

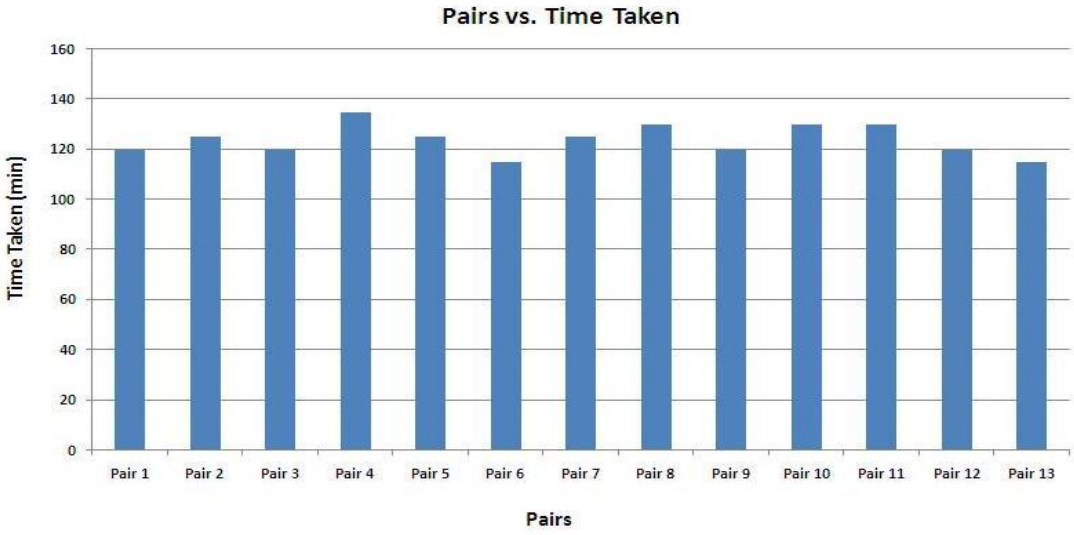


Figure 4.11 Time analysis for the pairs with respect to experiment 2

The result of the second experiment pairs with respect to lines of code and time taken to complete the assignment is shown in Figure 4.10 and Figure 4.11. Notice that there is only one unsuccessful pair as indicated in the Figure 4.10 because the pairs have been selected according to their skill level, Figure 4.10 also indicates that the number of unsuccessful pairs has reduced when compared to the first experiment. This is because of the fact that the pairs have been formed based on the graph matching method. Table 4.2 shows the lines of code and time taken for experiment 2.

4.6.3 Pairs chosen using weighted graph matching considering skill level

In the previous two experiments, the skill levels of pairs was not considered. But, a pair is successful in this kind of programming only if they are paired by considering their skill levels too, as suggested by Chaparro et al. (Chaparro et al. 2005). This was the reason for revamping the methodology in forming pairs and a third experiment was conducted, in which the edge weights incorporate the skill/knowledge level of the pairs. To arrive at the

perceived skill level of students, their lab marks, faculty rating and personal rating of the students were also included. The faculty rating of each student for C programming language and Data Structures were obtained from the respective faculty. The skill level SL_k of k[th] student is then derived as shown in Equation (4.3),

$$SL_k = L_c + L_d + F_c + F_d + P_c + P_d \quad (4.3)$$

where L_c and L_d represent the laboratory performance marks of the k[th] student for C Language and Data structures respectively taken from the semester examinations; F_c and F_d represent the scores given by the concerned faculty for C Language and Data structures respectively; P_c and P_d represent the self-evaluated scores given by the student for C Language and Data structures respectively. For this experiment, the final score S_k for k[th] student is calculated by adding the skill level SL_k from Equation (4.3) and QS_k from Equation (4.1).

$$S_k = QS_k + SL_k \quad (4.4)$$

To ensure that a novice is paired with a partner who has better domain knowledge and with a view to make the overall performance better, a different strategy in forming pairs was adopted. The student scores were sorted in descending and the students were split into two groups A and B according to the sorted scores. Group A included the first 13 students in the sorted list and group B comprised the rest. This means that on average, students in group A have higher skill levels when compared to students in group B. Unlike the student network as the complete graph K_{26} in the second experiment, here the network is a complete bipartite graph $K_{13,13}$. As in the second experiment, the weight of vertex k is the score S_k of k[th] student and the edge weights are calculated by averaging the corresponding vertex weights. The bipartite graph structure restricts that a student from

group A can be paired only with a student from group B, ascertaining that at least one among the pair has better skills in comparison with his/her counterpart. This ensures that all pairs are balanced in terms of skill level/domain knowledge. The bipartite graph with the newly computed edge weights is then fed to Hungarian graph matching algorithm to find a maximum weighted bipartite matching.

Table 4.3 Lines of Code and Time taken for Experiment 3

Pa Pairs	Lines of Code	Time Taken (In minutes)
Pair 1	80	100
Pair 2	82	82
Pair 3	83	83
Pair 4	85	105
Pair 5	90	106
Pair 6	100	106
Pair 7	105	107
Pair 8	110	90
Pair 9	111	120
Pair 10	120	118
Pair 11	123	120
Pair 12	130	106
Pair 13	132	120

The lines of code and time consumed by each of the pairs in the third experiment are shown in Figure 4.12 and Figure 4.13 respectively.

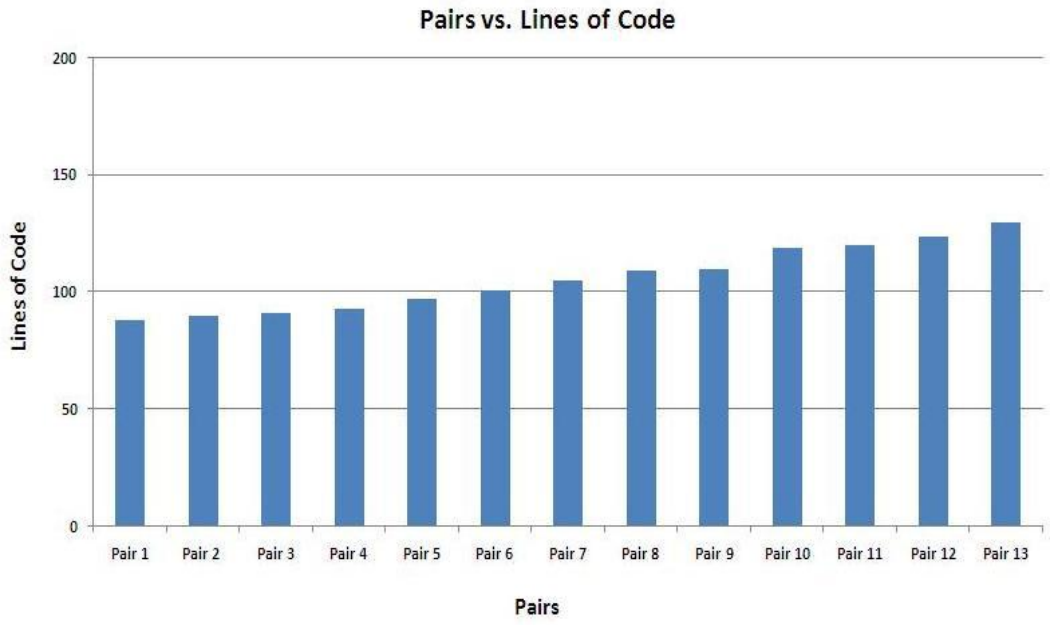


Figure 4.12 Lines of Code analysis with respect to experiment 3

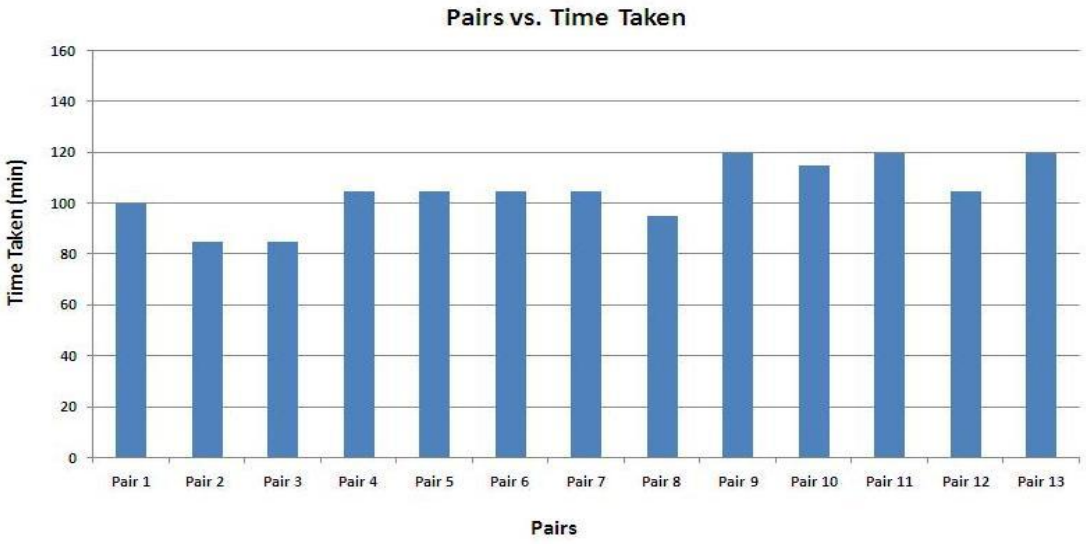


Figure 4.13 Time Analysis for the pairs with respect to experiment 3

It is to be noticed that the lines of code and time taken by the new pairs have considerably reduced and for this experiment, all pairs were successful. This method of pairing is best suitable when novices are present since it reduces the possibility of two beginners/proficient's getting paired together. Table 4.3 shows the lines of code and time taken for experiment 3.

4.7 RESULTS ANALYSIS AND DISCUSSION

In this section, different results of all the three experiments were compared and analysed in terms of Lines of Code and time consumed. Figure 4.14 and Figure 4.15 plot the graph of the average lines of code and average time consumed by all pairs for the three experiments.

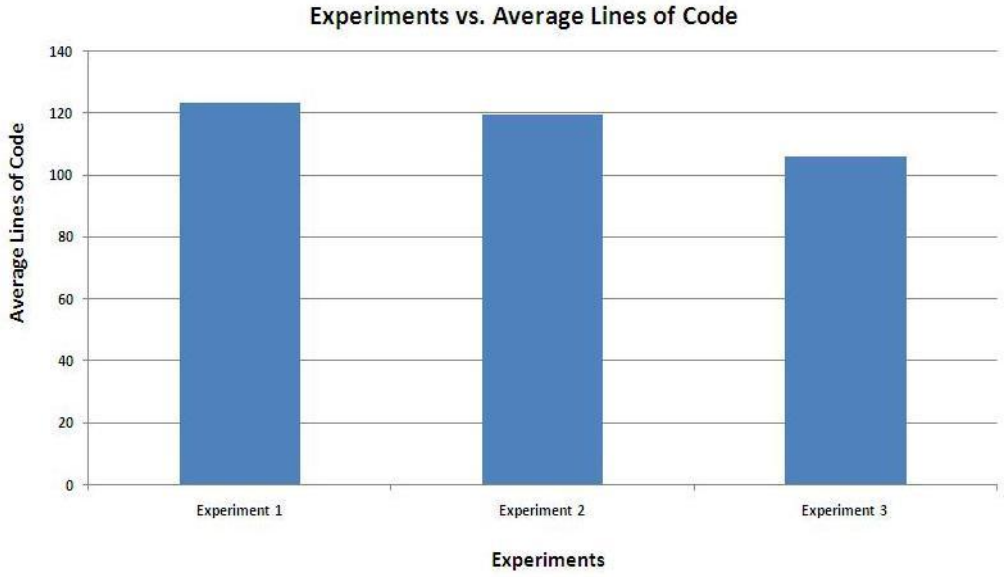


Figure 4.14 Average lines of code analysis for all three experiments

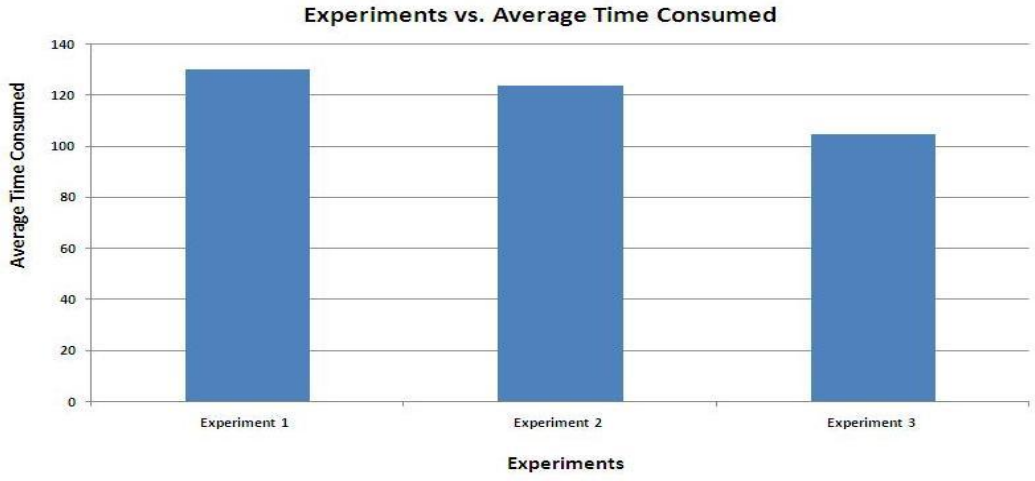


Figure 4.15 Time comparison Analysis for all three experiments

From Figure 4.14, it can be seen that the average lines of code for the second experiment is significantly lesser than that of the first experiment. But the average lines of code for the third experiment are significantly lesser when compared to the other two experiments. This is because in the second experiment, the pairs are selected by the graph matching algorithm but without considering their skill level and in the third experiment pairs are selected by graph matching algorithm by considering their skill level.

Also, Figure 4.15 depicts that the average time consumed by the first experiment pairs to complete their programs, is higher than that of the second and third experiment pairs. Of all the three experiments, the third experiment has the least time. This is due to the fact that the pairs selected for the third experiment were based on graph matching algorithm and their skill level was also considered while selecting the pairs.

Next the completion times of the pairs for the three experiments were also analyzed. The three time intervals (in minutes) considered here are 80-100, 101-120 and 121-140. It can be observed that the third experiment pairs have completed faster within 120 minutes, while many pairs have taken up to 140 minutes during first and second experiments. Figure 4.16 compares the number of pairs for each experiment and the time intervals for their programming.

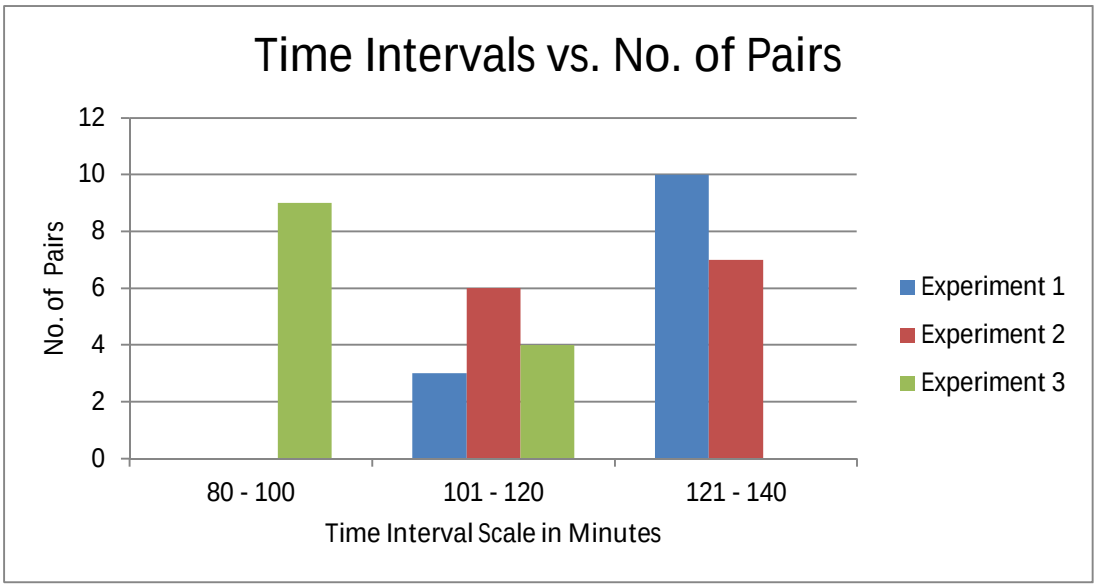


Figure 4.16 Time Analysis for the pairs in each of the experiments

The results indicate that the second and third experiments showed better performance than the first experiment. Especially, the third experiment in which the computation of maximum weighted matching included skill levels too, is experimentally found to produce fruitful outcome. This is due to the fact that, the pairs were selected based on their skill level and due weightage was given for the psychological factors to analyse the pair compatibility between pairs. The psychological factors of the pairs were analysed based on the scores obtained for the set of questions given in pair programming questionnaire-A in section 4.8.

After the completion of every experiment, feedback was obtained from all the students for their experience in pair programming. A questionnaire was distributed after each assignment was complete, to know how they felt about their new partners and to know if it was productive to them. Repeating the assessment of individuals was not done after this questionnaire. Students felt that they were more compatible with their partners of the third experiment than those of the previous experiments; their discussions with their new partners were fruitful and it filled the holes in their

subject knowledge. The latter feedback provides strong evidence that skill level plays an important role in pair programming. They opined, pair programming provided them mutual motivation and boosted their confidence in completing the task successfully, as the partners can help each other when required. They were even motivated to work in pairs again.

4.8 PAIR PROGRAMMING QUESTIONNAIRE – A

The questionnaire set A used for conducting experiments is given below. This questionnaire was distributed prior to the conduct of first experiment and this is used to evaluate the value of QS_k. The weightage value for each option is also given in the table 4.1 ranging from maximum value of 4 to minimum value of 1.

S.No.	Question with options
1	Do you know the domain of work? a) Yes b) No
2	If your answer is "Yes" to question number 1, how much do you think is your competency level? a) High b) Average c) Low d) Don't Know

S.No.	Question with options
1	Assume you are working in a pair, you find a mistake in your partner's code. How will you react to this situation? a) You will intimate your partner immediately and guide him/her correctly. b) You will wait until your partner finds the mistake by himself/herself. c) You will not bother about the mistake as you will rectify it during your turn. d) You will get your partner's reason for such a work, before pointing it as a mistake.

S.No.	Question with options
2	What will you do if your partner is unable to find a solution for a problem? a) Guide your partner to use resources like internet or books to find a solution. b) Ask your partner to find the solution by himself/herself. c) You will leave the problem as it is, because of other important activities. d) You rely on your own experience to find potential solutions to a problem and so you do not mind much even if your partner is not able to do.
3	You need to make an immediate decision. What will you do? a) You will consult your partner, discuss and then take a decision. b) You feel time will be wasted in unnecessary arguments during decision making and when making a decision, you trust your inner feelings and reactions, so you take the decision on your own. c) You feel that your partner is not capable of deciding, so you will make the decision by yourself. d) You avoid making important decisions until the pressure is on, so postpone it for a while.
4	How will you handle a stressful situation? a) You will seek the help of humour to reduce the tension around. b) You will be involved in a direct communication with your partner. c) You will lose patience with the need to get your partner involved in discussion. d) You will discuss to outsiders about the problems that you face in the team.
5	What will you do when conflicts arise between you and your partner? a) You will maintain silence for a while and avoid the arguments with your partner.

S.No.	Question with options
	b) You will stress for an honest discussion of the differences and the reasons for conflicts. c) You will explain and provide reasons to prove why one side is correct and the other is incorrect. d) You will try to break the tension with a supportive or humorous remark.
6	What will you do when things go wrong on the team? a) You will emphasize on listening, feedback, and participation. b) You will arrange for a candid discussion of your problems. c) You will work hard to provide more and better information. d) You will suggest revisiting your basic mission and start reworking on it.
7	Among the following, which do you think is difficult to do and that could put you in a troublesome situation? a) Questioning some aspect of your partner's work. b) Pushing the team to set higher performance standards. c) Working outside your defined role or job area. d) Providing your partner with feedback on their behaviour as a team.
8	Assume you and your partner know the domain of work and both are good at it. Which among the following, you think is required for pair problem solving? a) Co-ordination and co-operation by both members. b) High-level listening skills to absorb significant information. c) A willingness to interrogate your partner with tough questions during work. d) Acquisition of good, solid data that defines the problem well.
9	Suppose your partner doesn't have any knowledge in the domain in which you are going to work as a pair, what will you do? a) You will not accept that assignment. b) You will try to educate your partner by mutual reading.

S.No.	Question with options
	c) You will do the entire work by yourself and ask your partner to absorb what you are doing and thereby make him/her learn the area of work. d) You will give some time for your partner to learn and once he/she finishes, you both will commence the work together.
10	When your partner goes wrong at some point during the work, what will you do? a) You will criticize your partner so that he/she can learn from it. b) You will ignore your partner. c) You will specify the resources for your partner to learn. d) You will educate him by asking more specific questions and giving suggestions.
11	Suppose you find that your partner is more knowledgeable than you, what will you do? a) You will let your partner do the work and your involvement will be less. b) You will frankly admit to your partner and ask him/her to teach you. c) You will get some time for you to learn on your own. d) You will not take up such a project.
12	What will you do when pairing up with a new partner for a work? a) You will try to meet and get to know the person. b) You will ask direct questions about the goals and methods that you both need to work on. c) You will talk to your partner to know what is expected of you. d) You will engage in a discussion with your partner for clarity about your basic mission.
13	What according to you is the basis for the team decision? a) The team's mission and goals. b) A consensus of team members. c) An open and candid assessment of the issues. d) The weight of the evidence such as available information, statistics, etc.

S.No.	Question with options
14	How will you deal if you feel that your partner is too rigid? a) You will try to convince him/her with your own ideas. b) You will just ignore your partner for the time being and concentrate on your work. c) You will inform your superior for a better replacement. d) You will tell him frankly that he/she is going in a wrong direction.
15	If your partner keeps on finding faults with you and criticizes your approach, how will you react? a) You will try to explain and convince him/her about the effectiveness of your logic/approach. b) You will ignore his/her criticism as you firmly know your approach is right and you will not waste time on proving your point. c) You will inform your superior about your partner's attitude. d) You will tell him/her frankly that he/she is not encouraging.

Table 4.4 Answer Credits: Weightage for options in the order 4 3 2 1

Question Number	Weightage 4 3 2 1
1	d a b c
2	a d b c
3	a b d c
4	b a d c
5	b c d a
6	b a c d
7	c b d a
8	a c b d
9	b d c a
10	d c a b
11	c b a d
12	d b a c
13	c b a d
14	a d b c
15	a d b c

4.9 PAIR PROGRAMMING QUESTIONNAIRE – B

The questionnaire given after each experiment for feedback survey used for conducting experiments is given below.

Sl.No.	Question
1	Did your partner cooperatively follow the pair programming model (rotating roles of driver and navigator, questioning and making observations as the navigator)?
2	Did your partner contribute fully, fairly and actively, to the best of his or her ability, to the completion of the lab assignment?
3	Was your partner's participation, professional and cooperative overall?
4	If given another opportunity, would you like to work in pairs again?
5	Did you find social difficulties with your companion?
6	Do you think collaboration with your partner gives you more confidence in solving programming problems?
7	Do you this collaboration in the experiment will be more effective if it has more than two members?
8	Do you think pair programming process enriches your knowledge?
9	Assess the technical competency of your partner when compared to your competency : a) better b) about the same c) weaker
10	Assess how compatible you and your partner were a) Very much b) Good c) Not much.

4.10 CONCLUSION

Pair programming is definitely one of the best mutual teaching-learning methodologies when the pairs are compatible and has the drive to achieve maximum quality and productivity. The feedback survey conducted during our case study revealed that pairs of the third experiment were more comfortable and enthusiastic, which favors the significance of skill level in pairing. It was evident from the third experiment that the pairs were more compatible and they produced promising results. Essentially the success of pair programming depends on both the complexity of programming task and compatibility of the pairs. Only when the pairs are compatible with each other, the working environment will be more interesting for the pairs, which will improve their productivity. It will be a win-win strategy for both the partners where the job is expected to be completed successfully.

This work on pair programming using graph matching can be extended to distributed pair programming environment where the pairs will be geographically separated and the communication between the pairs can be through text chatting, voice or video conferencing. Moreover there is scope for enhancing the measure of compatibility by encompassing many other related factors.

CHAPTER 5

ENHANCING LEARNING EXPERIENCE OF E-LEARNERS IN LABORATORY COURSES USING PAIR PROGRAMMING

E-learning is a learning by utilizing electronic technologies to access educational curriculum outside of a traditional classroom. In most cases, it refers to a course, program or degree delivered completely online.

5.1 E-LEARNING

E-learning courses are the courses that are specifically delivered via the internet to somewhere other than the classroom where the professor is teaching. It is not a course delivered via a digital video disk (DVD), video tape or over a television channel. It is interactive in that one can also communicate with their teachers, professors or other students in their class. Sometimes it is delivered live, where one can interact in real time and sometimes it is a lecture that has been prerecorded. There is always a teacher or professor interacting /communicating with the e-learner and grades his/her participation by evaluating the assignments and tests. E-learning has been proven to be a successful method of training and education and is becoming a way of life for many students across the globe.

5.2 E-LEARNING IN LABORATORY COURSES

Laboratory courses constitute one of the core competencies that graduates from information systems discipline are expected to possess. Laboratory courses in e-learning are just a curricular formality without bothering about the learning experience. Lot of practice is required for e-learners for acquiring a good learning experience, for which motivation is an essential factor. Research has suggested that the lack of a formalized structure for laboratory courses may be one of the factors responsible for learners' negative impressions of e-learning and also for the high failure rate in e-learning. Ability to work in teams has been considered one of the most important learning outcomes of the laboratory courses.

This study highlights the importance of laboratory courses in e-learning and investigates whether the use of pair programming in laboratory courses would enhance the learning experience of e-learners. The final objective is to provide new learning experience to motivate e-learners and present laboratory courses as an easy and attractive challenge using pair programming. Experiments were conducted in Data Structures, problem solving and C programming courses. Results indicate that the learning experience of both the learners and teachers were improved in laboratory course and also showed an improvement in success rate.

5.3 RELATED WORK

In most of the e-learning systems, theory is given more importance. Practical lab assignments is not given in most of the e-learning systems. Learners of information systems courses cannot be trained with focus on theory only which is going to be forgotten with passage of time (Van Der Vyver& Lane 2003). Good programming skills are one of the core competencies that information system learners are expected to develop.

However, learners and teachers agree that learning laboratory courses is a hard through e-learning. Learners need to be adequately motivated in order to learn programming in a successful and effective manner. Learners will be motivated when they interact with other learners and/or teacher (Furberg et. al. 2013).

The main issue which may exacerbate e-learners' difficulties with laboratory courses is the lack of a formalised environment for collaborative peer learning (Preston 2005). Some of the challenges that e-learners face in laboratory courses may be overcome by allowing learners to collaborate with their peers. The pedagogical advantages of learner interaction in collaborative construction of knowledge are grounded in the social constructivist perspective of learning. Based on the constructivist pedagogical approach, actual learning takes place when students actively construct their knowledge through social interactions with their peers (Van Der Vyver 2003). Knowledge is discovered and constructed through communication and collective sense making. Collaborative learning benefits educators in computing domain. Engagement in collaborative activities causes individuals to master something that they could not do before the collaboration. Investigation of how collaborative learning can be used to enhance learning experience of e-learners in laboratory courses is the main objective. In e-learning, learners are located in geographically different locations; the current study employs a distributed collaborative programming technique referred to as distributed pair programming.

Distributed pair programming is a novel and successful collaborative paradigm used in software industry (Salleh et al. 2011). The idea is that two programmers work collaboratively on the same program from the different locations. One programmer is designated as the 'driver' and has control of the input devices. The other programmer is designated as the

'navigator' and has the responsibility of reviewing the code that has been typed to check for deficiencies, such as erroneous syntax and logic, misspellings and design issues (Braught, Wahls & Eby 2011). The navigator continuously examines the work of the driver, thinking of alternatives and asking questions. The driver and the navigator change roles frequently and different pairs are formed to facilitate the spread of information through an organisation.

It is the opinion of the industry experts that programmers working in pairs produce shorter programs with better design and fewer bugs than those working alone (Vanhanen & Lassenius 2007). This collaborative technique has also been successfully applied to the teaching of computer programming for beginners in classroom and a wide range of benefits have been reported, such as improved quality of code, decreased time to complete, improved understanding of the programming process, enhanced communication skills and enhanced learning (Salleh et al. 2011;Williams & Upchurch 2001). In pair programming, one learner will follow another learner and will try to imitate. One learner will be the camaraderie of another learner. It is found that mathematical logic skills were enhanced when pair programming practice was followed. It is also proved that collaborative learning enhanced student experience in producing Wiki websites (Tsai et. al. 2011). Pair Programming had positive effects on student engagement and performance within computer science lectures (Maguire & Maguire 2013).

5.4 THE INTRICACIES OF LEARNING LABORATORY COURSES IN E-LEARNING ENVIRONMENT

Computer programming laboratory courses may be viewed as a method for some problem solving. Knowledge transfer is expected to be easier if the prior knowledge and/or experience of the learners are similar to

the knowledge transfer being done. Programming laboratory courses are greatly enhanced through learner-learner interaction. Where there is problem in this knowledge transfer such as an error which the learner cannot explain, overcoming that problem is faster through minimised distance between teaching and learning (Denner et. al. 2014). Laboratory courses are not similar to other courses. There is some uniqueness to laboratory courses in e-learning that must be considered when one contemplates an ideal environment for learning programming courses. Teaching and learning programming courses have their intricacies as well as problems not fully overcome (Carver et. al. 2007).

Online learning has major benefits but when programming is taught online, another set of concerns must be considered. Additionally the benefits derived from an online environment for different courses differ. International Data Corporation reports that enrolments in e-learning courses are growing at 33% a year and will continue to climb. Most of the e-learning systems provide Virtual Learning Environment (VLE) and Integrated Development Environment (IDE) to offer laboratory courses. There are new ways in learning programming language as such using virtual learning environments, evolving programming environments and software programs and applications.

The technology exists for this application providing for computer-based instruction or asynchronous and synchronous learning networks. Virtual Learning Environment (VLE) is a set of teaching and learning tools intended to develop a student's learning capability via computers and the Internet in the learning process (Rosenberg, 2001). Learners with low motivation or bad study habits may fall behind.

5.4.1 Research Overview and Hypothesis

Much of the research on distributed pair programming as a pedagogical technique has focused on the teaching of introductory programming courses to beginners, with fewer studies investigating its applicability for expert programmers. Also, while there is a growing body of research in the area, more studies have focused on pair formation and its effectiveness. The outcomes of adopting a collaborative pair programming paradigm for enhancing learning experience of e-learners in laboratory courses is described here.

Comparison is done with the learning efficacy of e-learners in laboratory courses with pair programming and without pair programming. The continuity of the learner cohort permits analysis of various outcomes of the pedagogical intervention, such as learning experience and efficacy. Thus, the following hypotheses are formulated about the e-learners in laboratory courses using pair programming in the improvement of learning experience and efficacy:

H1 **:** The e-learners who use pair programming will have better learning experience than those who do not use it.

H2 **:** The e-learners who use pair programming in laboratory courses will obtain higher grades than those learners who do not use it.

H3 **:** To establish whether learners benefit from a peer programming intervention in terms of their academic performance in both continuous assessment and examination results,

H4 **:** Dropout in e-learning will be decreased because of the satisfaction level of e-learners.

5.4.2 Methodology

Experiments were conducted with the e-learners of our e-learning system taking Data Structures laboratory and problem solving and C programming laboratory courses. These two laboratory courses are offered at both UG and PG level in science and engineering stream. Table 5.1 shows the total participants and the same is shown as chart in Figure 5.1.

Table 5.1 Experimental Analysis

	Name of the Laboratory Course	Gender		Stream				Total
		Boys	Girls	Science		Engineering		
				CS	Non-CS	CS	Non-CS	
UG Level	Problem Solving and C programming Lab	180	114	74	47	116	57	294
	Data Structures Lab	152	97	87	30	91	41	249
PG Level	Problem Solving and C Programming Lab	45	92	58	19	43	17	137
	Data Structures Lab	41	67	53	21	18	16	108

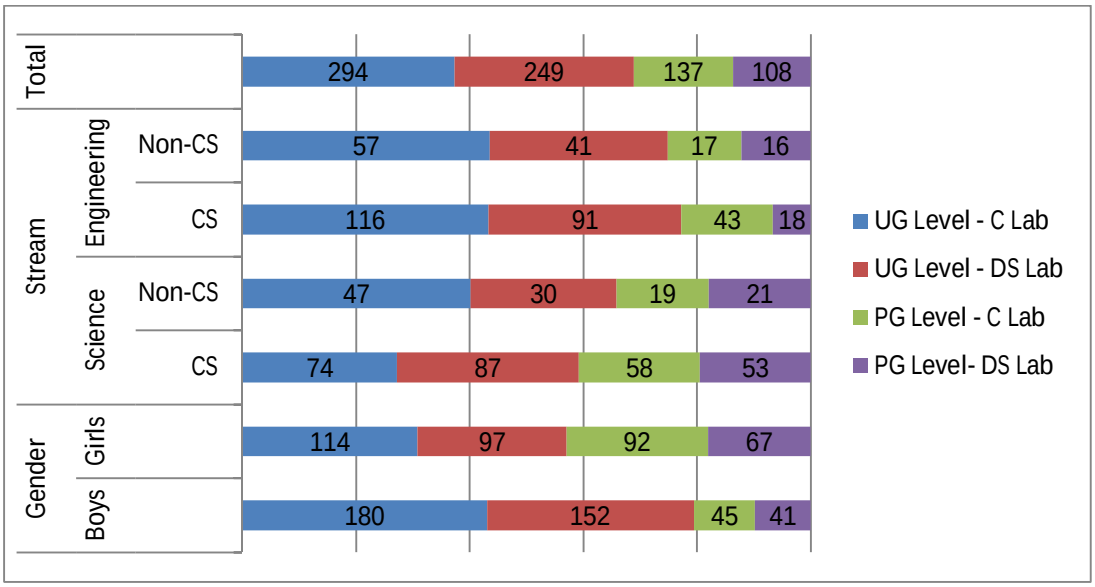


Figure 5.1 Total number of participants in the experiment

5.4.3 Experimental Design

A quasi-experimental design was employed with discipline of study, level of study and gender as the key independent variables. Students' performance in lab examination was the key dependent variable, but measures of programming confidence, perceptions of the pair programming intervention, dropouts were also examined.

5.4.4 Experimental Procedure

The experiment was conducted in an educational institution. Students' motivation, learning experience and satisfaction were analysed by means of observation and satisfaction questionnaires. In addition it was necessary to analyse the effects of the system on students' academic outcomes and dropout of students. For this, the experimental research method was applied (Oncu & Cakir 2011) and experimental and control groups were established in order to identify a relationship between variables. In order to evaluate the experience all students took the two courses during the academic year 2013–2014 were considered as belonging to the experimental group. Their academic results and dropouts would then be compared with those obtained by the students taking the same courses during the next academic year. The following principles were established:

- A total of 788 students with different gender who were enrolled in the two lab courses in UG / PG level, participated in the experiment.
- Necessary training was provided to the students to use e-learning system.

5.4.5 Instruments and data collection

Two instruments were used in this study: a) the students' final examination grades in the courses for both academic years, and b) a survey, which measures students' satisfaction with their learning experience using the system. Specifically, the survey was composed of three different parts:

- Personal data for statistics: age, gender, computer skills.
- Five-score Likert-type scale items, which ranged from "Strongly agree", "Agree", "Neutral", "Disagree" and "Strongly disagree", with a score ranging from 5 to 1 respectively for analysing the level of satisfaction.
- Yes/no items for assessing both the quality of the problems posed and the functionality of the on-line Judge.

Data from the survey was collected on-line when the courses finished. The survey was completed by all the students. The data collected on students' performance which was the final examination grades were analysed for group comparison using the Student T-Test. This statistical measure indicates whether the means of two groups are statistically different from each other in order to be able to compare them. In addition, in order to check whether students' satisfaction differed according to gender, subject of study and UG/PG level and to investigate whether there was any interaction among these variables, a two-way analysis of variance (ANOVA) was also conducted.

5.5 RESULTS AND DISCUSSION

In this section the results are analyzed and discussed to investigate whether the e-learning process could be improved using the pair programming technique.

5.5.1 Comparisons between time spent on learning and academic performance

Table 5.2 outlines the overall Mean and Standard Deviation (SD) of the scores for total time spent on learning, and final assessment across the two academic years. A series of dependent T-tests revealed a significant difference between the two groups on any of these measures ($p > 0.05$).

Table 5.2 Mean, Standard Deviation (SD) and t test of the scores for total time spent on E-learning and final assessment mark across the two academic years.

	Academic year 2013 - 2014 (without pair programming)		Academic year 2014 – 2015 (with pair programming)		t test	
	Mean	**SD**	**Mean**	**SD**	**T**	**P**
Total time spent on the system	162.45	2.0256	276.87	0.819	2.017	0.030
Final exam result	52.59%	1.383	87.37%	0.753	2.142	0.025

A sample of 788 students in the academic year 2013-2014 was taken for the experiment and an analysis was done on the total time spent by the students on the system. The average time spent by the students in the academic year 2013 – 2014, without pair programming was 162.45 hours and average time spent by the same set of students in the academic year 2014-2015 with pair programming was 276.87 hours.

The final examination was conducted and it is observed that the average mark scored by the students was 52.59% with a standard deviation of 1.383. This is in the case of students in the academic year 2013-2014, where pair programming was not employed for the e-learning system. The average mark scored by students in the final examination in the academic year 2014-2015, where pair programming was employed is 87.37% with a standard deviation of 0.753.

As the sample taken was dependent, we consider the student t test to test the hypothesis .

H0 **:** The e-learning was effective when it is employed with pair programming during the academic year 2014-2015.

H1 **:** The e-learning was not effective when pair programming was not employed during the academic year 2013-2014.

From the statistical analysis, it is observed that there is an increase in the average marks and a decrease in standard deviation when we compare with the academic year 2013-14 without pair programming and with that of academic year 2014-15 with pair programming. The computed P values for total time spent on the system and for final exam result were observed to be 0.030 and 0.025 respectively which are less than P value where P=0.05 (Level of Significance). Hence H0 is accepted.

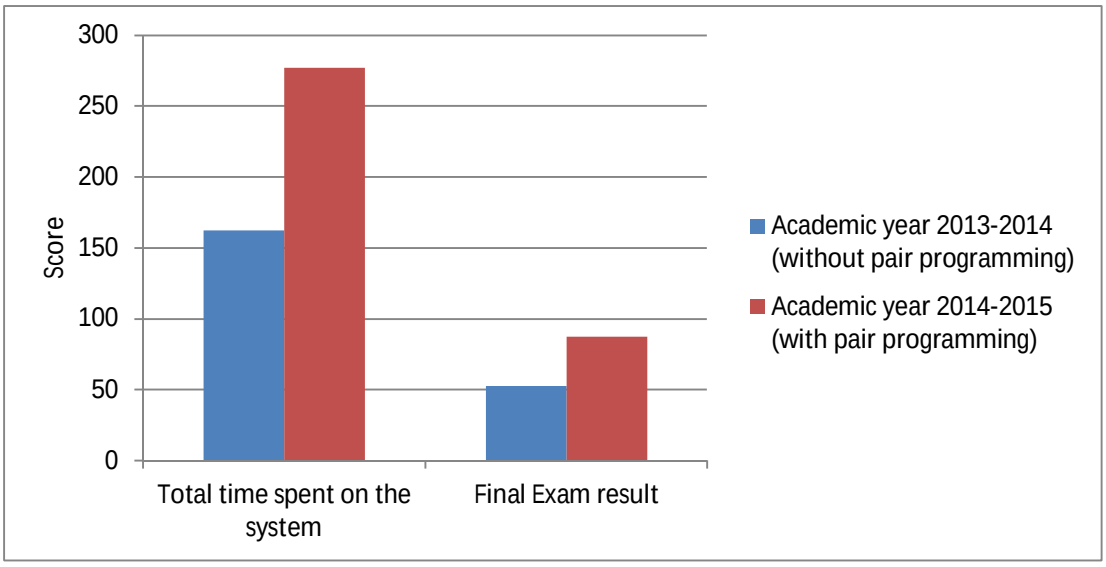


Figure 5.2 Time comparison Analysis on learning and academic performance

An improvement in the final score can be observed when pair programming is applied to learning programming using e-learning. According to the results in Figure 5.2, students who used pair programming achieved significantly better academic outcomes than those who did not use it across all courses. This result also shows that students are more interested in spending more time in learning. This result indicates that the hypothesis H1 is supported. The trend, as seen in Table 5.3, clearly indicates that as the semester progressed, the students showed more interest in learning programming and the dropout rate is reduced. This also shows that for students having no programming background, the maximum learning experience came from the lab work. By the end of the semester, all students performed well in the final examinations.

5.5.2 Comparisons between dropout rate and failure rate

The overall failure rate and dropout rate across the two academic years is shown in Table 5.3. Failure rate is the average number of students who failed in the final examination. Dropout rate is the average number of students who attended the course and did not appear for the final examination.

We observe that the failure rate in case of the students in academic year 2013-2014 where pair programming is not employed (0.317), is more than that of the failure rate of students in the academic year 2014-2015, where pair programming was employed (0.048). The dropout rate is also more (0.108) where pair programming was not employed than that of the students in the academic year 2014-2015 where pair programming was employed (0.019).

Table-5.3 Overall failure rate and dropout rate across the two academic years

	Academic year 2013 - 2014 (without pair programming)	**Academic year 2014 – 2015 (with pair programming)**
Failure rate	0.317	0.048
Dropout rate	0.108	0.019

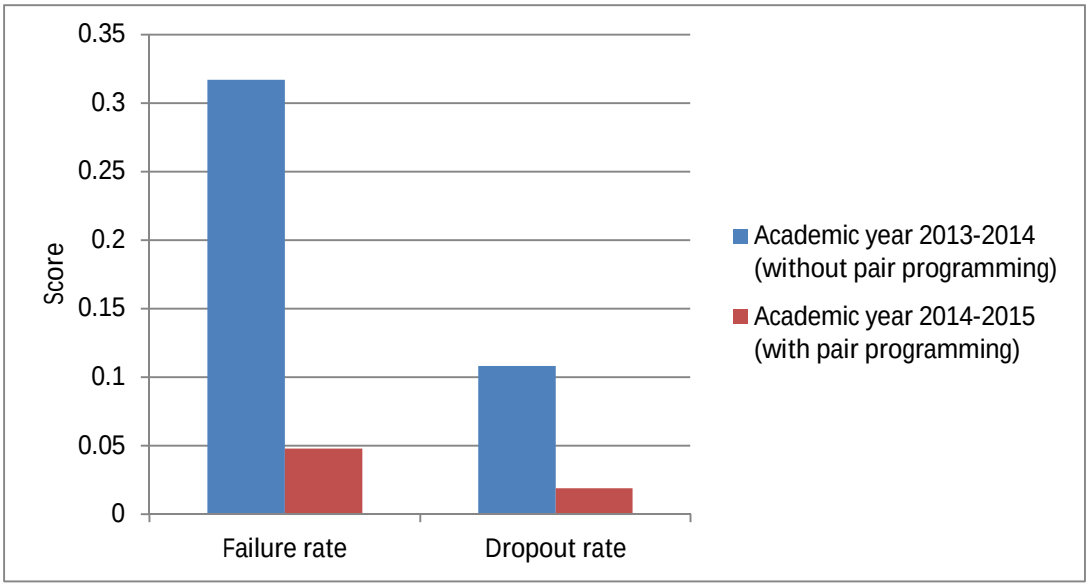


Figure 5.3 Overall failure rate and dropout rate across the two academic years

The major problem in e-learning is the lack of confident and motivation in learning. It can be observed that when pair programming is applied to learning programming laboratory courses in e-learning the overall failure rate and dropout rate is significantly reduced. According to the results in Figure 5.3, learners who used pair programming are confident in completing the course successfully. This gives a positive sign for the universities and organizations that uses e-learning system.

5.5.3 Analysis of the satisfaction of students

In this section we analyse the students' degree of satisfaction with the use of pair programming in learning programming in e-learning system based on the survey data. The purpose of this analysis is to validate the usefulness of the system, since several studies (Donohue & Wong, 1997; Levy, 2007) suggest that students' satisfaction and motivation are important factors in measuring the success or effectiveness of the e-learning process. The analysis of results is done in general terms and also answering the research question and testing the hypotheses formulated.

Once the results of the surveys were available, their reliability was analysed. Cronbach's alpha was tested and the calculated alpha value for the learning experience of e-learners in programming laboratory courses was 0.95, indicating very high reliability (Straub 1989). In general terms, the survey data shows the learning experience was evaluated positively by students. Figure 5.4 summarises the survey results for each course in UG level, where bars represent the average score assigned to each item (5 being the maximum score). Figure 5.5 summarises the survey results for each course in PG level. It is also clear that the students think that it helped to achieve academic excellence in learning goal. Most students reported a high learning experience with the pair programming.

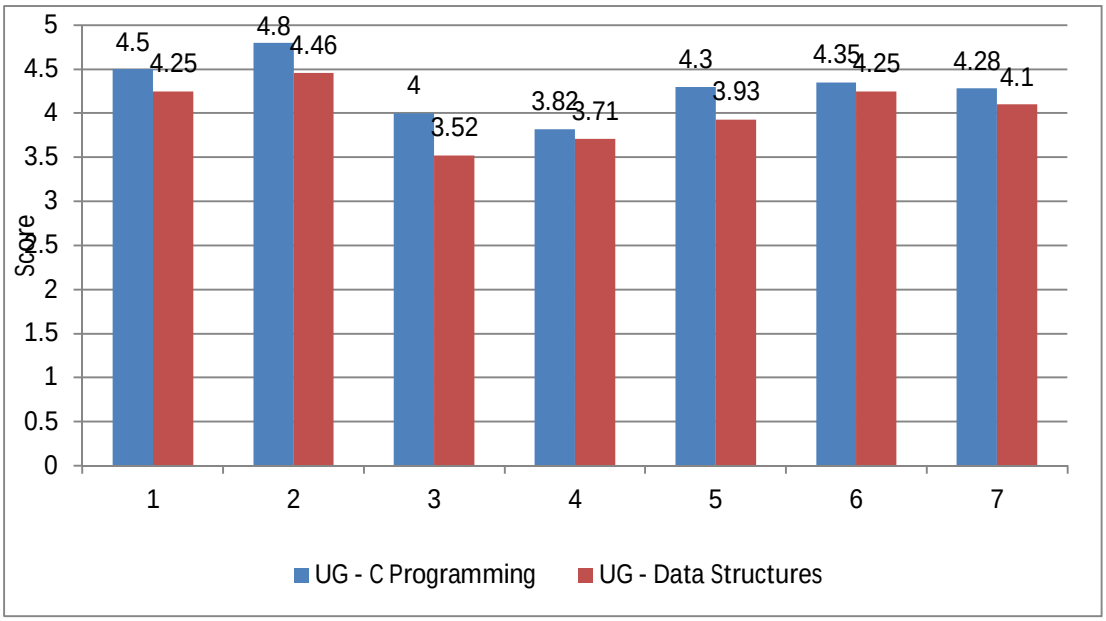


Figure 5.4 Survey results for UG students

In Figure 5.4 the index 1 in the horizontal axis represents Satisfaction Level of students in the e-learning system, index 2 represents, Importance of Lab courses in e-learning environment, index 3 represents participation in learning, index 4 represents Involvement in learning, index 5 represents Improvement in academic performance, index 6 represents, enabled to attain meaningful learning goal and index 7 represents Confidence in subject.

In Figure 5.5 the index 1 in the horizontal axis represents Satisfaction Level of students in the e-learning system, index 2 represents, Importance of Lab courses in e-learning environment, index 3 represents participation in learning, index 4 represents Involvement in learning, index 5 represents Improvement in academic performance, index 6 represents, enabled to attain meaningful learning goal and index 7 represents Confidence in subject.

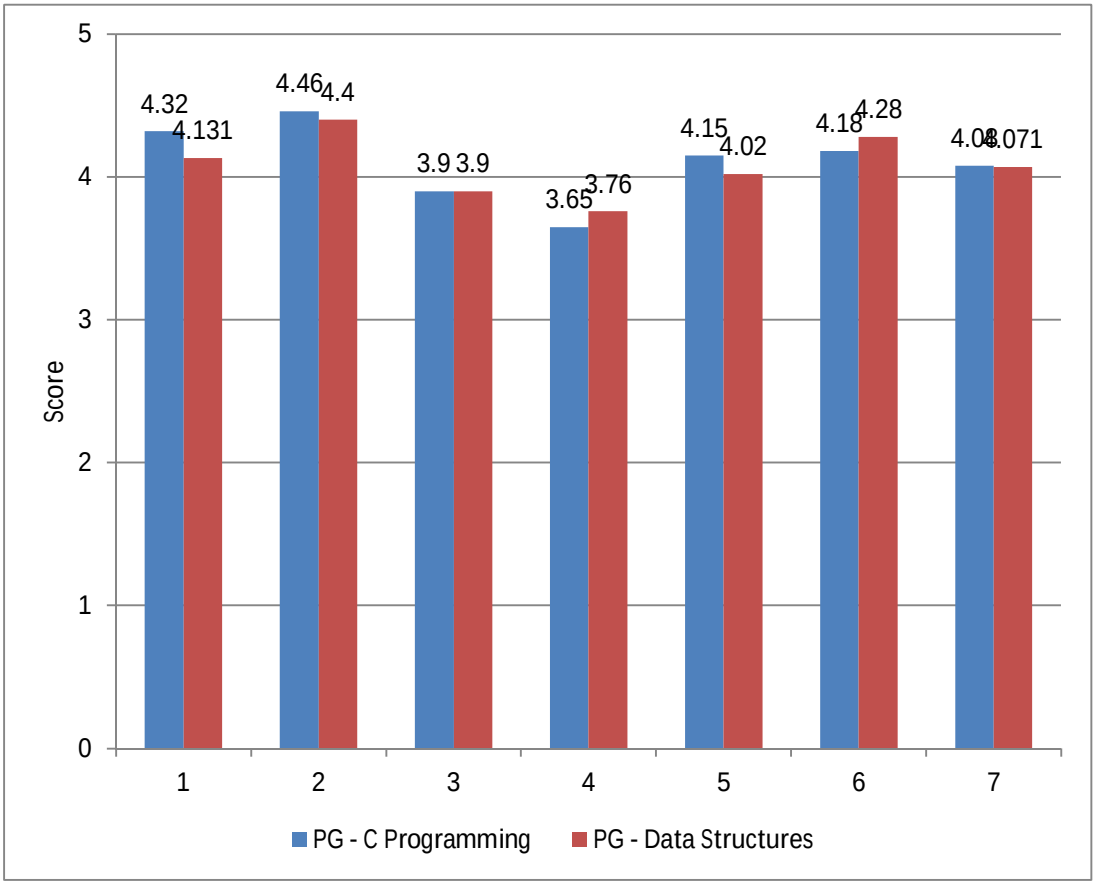


Figure 5.5 Survey results for PG students

Table 5.4 Mean and Standard Deviation (SD) of Learner's opinions regarding learning experience

Survey items regarding learners' learning experience	Mean	SD
Satisfaction Level	4.301	0.725
Importance of Lab courses in E-learning environment	4.524	0.779
Participation in learning	3.828	1.018
Involvement in learning	3.75	0.952
Improvement in academic performance	4.103	0.871
Enabled to attain meaningful learning goal	4.271	0.729
Confidence in subject	4.132	0.785

Table 5.4 shows a more detailed statistical study (with the mean and the standard deviation) of the different items in the opinion survey. The average value of scores obtained for the satisfaction level of students in the E-learning system employed with pair programming is 4.301 with a standard deviation of 0.725. When the students were asked about the importance of lab courses in e-learning environment, the average score was 4.524 with a standard deviation of 0.7779. It indicates that majority of students felt that lab courses in e-learning environment employed with pair programming was more effective than compared with a e-learning system without pair programming. The average score for students participation in learning, was 3.828 with a standard deviation of 1.018. The average score for involvement in learning, the average score was 3.75 with a standard deviation of 0.952. The average score for improvement in academic performance was 4.103 with a standard deviation of 0.871. When students were asked whether the e-learning system employed with pair programming enabled them to attain meaningful learning goal, the average score was 4.271 with a standard deviation of 0.729. The average score for gaining confidence in subject was 4.132 with a standard deviation of 0.785.

The results indicate that the satisfaction level of the students and learning experience in laboratory courses in e-learning environment that uses pair programming was very effective.

Table 5.5. Two-way ANOVA for the learner's satisfaction

	Sum of squares	**Degrees of freedom**	**Mean square**	**F value**	**p value**
Gender	4.919	1	4.919	4.88	0.037
Level of Study	3.634	1	1.817	1.8	0.184
Gender x Level of Study	0.378	1	0.189	0.19	0.828
Error	28.229	785	1.008		
Total	37.159	788			

Finally, the different hypotheses proposed about the relationship between the level of satisfaction and gender and level of study had to be validated. Since two factors namely, Gender and Level of Study were considered for the hypotheses, two way ANOVA table was considered for the statistical analysis.

H0 **:** No significant difference in the learning experience with respect to gender

H1 **:** There is a significant difference in the learning experience with respect to gender.

H3 **:** There is no significant difference in the learning experience with respect to level of study

H4 **:** There is a significant difference in the learning experience with respect to level of study.

H5 **:** There is no significant difference in the learning experience with respect to gender and level of study considered together..

H6 **:** There is a significant difference in the learning experience with respect to gender and level of study considered together.

Results of Table 5.5 indicate that students' learning experience was not different in relation to gender ($F=4.88$, $p> 0.05$). Hence H0 is accepted.

There is a significant difference in the learning experience with relation to level of study ($F = 1.8$, $p > 0.05$) or the interaction of both ($F = 0.19$, $p > 0.05$). Hence H3 and H5 is accepted in both of these cases.

Based on the results shown in Table 5.4 and the responses and statements of the learners, some of the evident advantages of pair programming that we could bring out effectively in our e-learning laboratory course were:

Collaboration and confident building: Studies have shown that pair programming creates an environment conducive to more advanced, active learning and social interaction, leading to students being less frustrated, more confident, and more interested in IT (McDowell et. al. 2006). Students who work in pairs tend to produce programs of higher quality and have higher course passing rates (Nagappan et. al. 2003) even when students pair program in a distributed manner. It has improved the team work quality among learners. Learners feel the fact that paired programmers were more comfortable in clearing their doubts with their partners. When they worked in pair, learners showed the confidence in learning the subject. They were able to state when something was right and the ability to admit when something was wrong. Another advantage that was found in the students' responses was that paired learners developed the tendency to work together even outside the class.

Learning efficacy: According to (Bevan et al. 2002), pairs spend less time working on assignments than individuals. In our experiment also inexperienced pair programmers could produce code of the same quality in the same time as experienced-solo programmers. Although paired programmers had to write more code, (individual and combined tasks), they seldom took more than an hour to complete the task. This happened because when students attacked the combined task, the students working in pairs could work out the logic much easily and in less time as they had already grasped the concept while working on the individual tasks.

Skill development: Collaborative programmers talked, discussed, and argued more than the individual programmers. They had the additional and increased opportunity to learn by watching how their partners approach a task, how they use programming language features, and how they use the development tools (Williams et al. 2000). They had the opportunity to better understand someone else's view by understanding how an issue looks from their partner's perspective. At such times, drawing from each person's unique talents and experience, a process known as 'pair brainstorming' occurs resulting in highly effective problem solving. The simple act of explaining an issue often leads to the solution faster.

Quality in learning: Knowledge is constantly shared between pairs (Jason, 2004). Though no specific measure were made about program defects, the instructors felt that compared to earlier batches when such a pairing was not tried out, the quality of the programs produced by learners improved significantly.

5.1 CONCLUSION

While there is much research to suggest benefits of pair programming (McDowell et al. 2003 ;Preston 2005;Salleh et al. 2011;Williams & Upchurch 2001) the current study is focused on using pair programming for laboratory courses in e-learning. This chapter reports on a study using the pair programming for the programming laboratory courses in e-learning teaching-learning process in four laboratory courses offered at UG and PG level. This approach has resulted in benefits such as enhancement of problem solving skills, efficiency, quality, trust, and teamwork skills. It has been also observed that paired laboratory experience is especially advantageous to e-learners. A hidden advantage that was evident from the learners' responses was that learners were motivated to work collaboratively even for other tasks.

Firstly, learners like this approach since they regard it as useful, facilitating the learning process, enabling to attain the learning goal and good learning experience. Moreover, the results of this study indicate that the use of pair programming in e-learning has important effects on the learners' academic outcomes and also the dropout rate is also reduced. The learners were motivated and involved in laboratory courses that created a confident in them. The learners obtained better final grades. Therefore, the results hereby presented suggest that this system can support effective learning strategies for laboratory courses in e-learning. The research showed several benefits of using pair programming in laboratory courses in e-learning such as enhanced learning, greater confidence in work quality, higher problem solving skills, enhanced interaction skills, and improved team building skills. The result also shows that e-learners had a good learning experience. The study also indicated several areas for future research.

Future studies can examine the effects of pair forming and automatic formation of pairs. Future studies can examine the use of pair programming in Non-Computer Science curriculum.

CHAPTER 6

PAIR RECOMMENDER SYSTEM : AN ASSOCIATION RULE BASED APPROACH

A pair recommender system based on association rule mining approach is devised. In Data mining, Association Rule Mining (ARM) is a technique to discover frequent patterns, associations and correlations among item-sets in data repositories. Association among programmers are found using association rule mining which measures the pair compatibility between the programmers.

6.1 INTRODUCTION

ARM and Apriori algorithm to solve ARM problems was introduced by (Agarwal et al. 1993). Association rules are used in many areas like market basket analysis, social networks, stock market etc. Here we use association rules to discover compatibility between pairs. Pair compatibility is influenced by various parameters like skill level, technical competence, designation, experience, personality interests, time management, learning style and self esteem.

A database in which an association rule is to be found is viewed as a set of tuples. In market basket analysis a tuple could be {bread, butter, jam}, which is the list of items purchased by a customer. Association rules are discovered based on these tuples and it represents a set of items that are purchased together. For example the association rule {bread} ⇒ {butter,jam}

means that whenever a customer purchases bread he/she will purchase both butter and jam in the same transaction. In the pair compatibility context, each tuple represents a parameter and the list of programmers satisfying the parameter. For example, the tuple {A,B,C} may be the list of programmers having the skill of developing programs using java programming language. Each tuple in the database represents a parameter with the list of programmers satisfying the same. An association rule found on this dataset may be of the form $\{A\} \Rightarrow \{B,C\}$ which means the programmer A can be paired with programmer B as well as with programmer C.

Experiments were conducted by selecting the pairs from the recommendation based on association rule and for solo programmers. For the elective course “Semantic Web”, students have to develop an application identified as a package. They have to do two packages identified as package1 and package2 and initially package1 was given to the selected pairs chosen based on association rule and it was evaluated. Again package2 was given by changing the pairs selected from the recommendation based on association rule. The results show that the productivity was almost the same for the two packages. Again package1 was given to the solo programmers and evaluated. Experimental results show that there is 23% increase in LOC and 70% in time when compared with pair programming of package1 and there was an increase of 32% in LOC and 53% in time when compared with pair programming of package2. The main advantage of the proposed method is recommendation of more than one pair is possible. This gives more freedom in choosing the pairs based on the complexity of the problem. Also, if a pair breaks due to unforeseen reasons the project can be still continued with next pair. This ensures the completion of the project.

6.2 RELATED WORK

The major factors to be considered in pair programming are product quality and productivity rate. The current research is focusing on influence of these factors on pair compatibility. The impact of pair compatibility was studied by (Katira et al. 2004), in which, they observed that the pairs with same skill had significant influence on pair compatibility. Further they also studied the impact of personality types, self-esteem and technical competence. In their improved work they analysed the impact of gender and ethnicity. Finally they conclude that pairs with same skill are compatible when compared with other factors. (Laurie Williams et al. 2003), extended this work and considered additional factors like time management, learning style and work ethic. They conclude that students prefer to pair with someone they perceive to be of similar technical competence. Their results show that pairing specific learning style, yields very compatible pairs, potentially because of their ability to complement each other's expertise. They further conclude that pairing students with strongly dissimilar work ethics will more likely yield to incompatible pairs.

Individual programming abilities are measured by assessment strategies framed by (Jan Hendrik et al. 2009). (Tomayko et al. 2002) conclude, that during pair programming, the number of defects produced were very less. (VanDeGrift et al. 2004) analysed that pair programming increases the programming performance and confidence. In a recent study, (Sultan Alshehri et al. 2014) use Analytic Hierarchy Process (AHP) to decide the best pairs in pair programming. From the studies, they found that the expert-expert is the best pair, other personalities and factors could play significant role that may need efforts to compromise these factors and add them to the criteria to be ranked and evaluated.

6.3 ASSOCIATION RULE MINING

Association rule mining is a method for discovering important relations between variables in large databases. It is widely used in industries to identify strong rules discovered in the data using different measures of association. These rules are generated by analyzing data for frequent conditional patterns to identify the most important relationships within the data. These rules will help to uncover relationships between seemingly unrelated data and predicting parametric behavior. These methods are applied in different industries like retail, banking, and e-commerce specifically for shopping basket data analysis, product clustering etc.

6.3.1 Association Rule

The problem of association rule mining is defined as: Let I = $\{i_1, i_2,...,i_n\}$ be a set of items. Let T = $\{t_1, t_2,\ldots, t_m\}$ be a set of transactions. A rule is defined as an implication of the form $X \Rightarrow Y$ where, $X, Y \subseteq I$ and $X \cap Y = \emptyset$, where X antecedent or left-hand-side (LHS) and Y consequent or right-hand-side (RHS).

For example consider the set of transactions shown in Table 6.1. An association rule $\{i_3\} \Rightarrow \{i_2\}$ implies that a customer purchasing an item i3, will also purchase the item i_2.

6.3.2 Quality measures

The quality measures to measure interestingness of association rules are support, confidence and lift. The measure support is an important measure because a rule that has very low support may occur simply by chance. Support is often used to eliminate uninteresting rules. Support also has a desirable property that can be exploited for the efficient discovery of

association rules. The measure confidence measures the reliability of the inference made by a rule. For a given rule $X \Rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence also provides an estimate of the conditional probability of Y given X. The measure lift computes the ratio between the rule's confidence and the support of the item-set in the rule consequent. Lift is simply the ratio of these values: target response divided by average response. The measures Support, Confidence and Lift are defined as below :

6.3.3 Support

The support value of an item-set X with respect to T, where T is the set of transactions, is defined as the proportion of transactions in the database which contains the item-set X and denoted as supp(X).

For example consider the set of transactions in Table 6.1. The item-set X = $\{i_2, i_3\}$ has a support of 0.5 since it occurs in 2 out of 4 transactions.

6.3.4 Confidence

The confidence value of a rule, $X \Rightarrow Y$, with respect to the set of transactions T, is the proportion of the transactions that contains X which also contains Y. Confidence value of a rule $X \Rightarrow Y$ is denoted as $conf(X \Rightarrow Y)$ and defined in the Equation (5.1).

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{5.1}$$

For example, consider the following transaction database T containing items I=$\{i_1, i_2, i_3, i_4, i_5\}$ as shown in Table 6.1.

Table 6.1 Transaction database

TID	Item-sets
T100	i_1 i_3 i_4
T200	i_2 i_3 i_5
T300	i_1 i_2 i_3 i_5
T400	i_2 i_5

Consider the association rule $\{i_3\} \Rightarrow \{i_2\}$. Confidence for this association rule is given below.

$$conf(\, i_3 \Rightarrow i_2 \,) = \frac{supp(i_3 \cup i_2)}{supp(i_3)}$$

$$= \frac{2}{3} = 0.67$$

In the above example as shown in Table 6.1, item-set i_2 and i_3 occurs in occurs in two transactions, the value of $supp(i_3 \cup i_2)$ is 2. Since the item-set i3 occurs for transactions T100, T200 and T300 the value of $supp(i_3)$ is taken as 3.

6.3.5 Lift

The lift of a rule is defined as the ratio of the observed support to that expected if X and Y were independent. Lift value of a rule $X \Rightarrow Y$ is denoted as $Lift\,(\, X \Rightarrow Y \,)$ and defined as in the Equation (5.2).

$$Lift\,(\, X \Rightarrow Y \,) = \frac{supp(\, X \cup Y)}{supp(X)\; supp(Y)} \qquad (5.2)$$

Consider the example shown in Table 6.1.

$$Lift\ (i_3 \Rightarrow i_2\ ) = \frac{supp(i_3 \cup i_2)}{supp(i_3)\ supp(i_2)}$$

$$= \frac{2}{3\ .\ 3} = 0.22$$

In the above example as shown in Table 6.1, the item-set i_3 and i_2 occurs in T200 and T300 and hence the value $supp(i_3 \cup i_2)$ is taken as 2.

Since the item-set i_3 occurs 3 times, the value of $supp(i_3)$ is taken as 3 and value of $supp(i_2)$ is also 3 since item-set i_2 occurs 3 times shown in Table 6.1.

6.4 APRIORI ALGORITHM

One of the most popular algorithms for mining association rules is Apriori algorithm devised by Agarwal et al. (1993). It is used to extract frequent item-sets from database and getting the association rule for discovering the knowledge. A common strategy of association rule generation is usually to decompose the problem into the following two phases.

- Frequent item-set generation, whose objective is to find all the item-sets that satisfy the minimum support threshold. These item-sets are called frequent item-sets.
- Rule generation, whose objective is to extract all the high confidence rules from the frequent item-sets found in the previous step. These rules are strong rules.

This strategy of association rule generation will help to find frequent item-sets and to extract all high confidence rules.

6.4.1 Frequent item-set Generation

Consider the set of all items I={a,b,c,d,e} in some data set. A lattice structure can be used to enumerate the list of all possible item-sets and is shown in Figure 6.1. The item-set {a,b,c,d,e} is the super set of all other item-sets. For every subset and its superset a line has been drawn. This resembles a lattice structure. From Figure 6.1, it is easy to identify all the subsets of a superset and vice-versa.

In general, a data set that contains *k* items can potentially generate up to $2^k - 1$ frequent item-sets, excluding the null set. A brute-force approach for finding frequent item-sets is to determine the support count for every candidate item-set in the lattice structure. To do this we have to compare each candidate against every other transaction. Such an approach is very expensive, because it requires O(NMw) comparisons, where N is the number of transactions, $M=2^k-1$ is the number of candidate item-sets and w is the maximum transaction width.

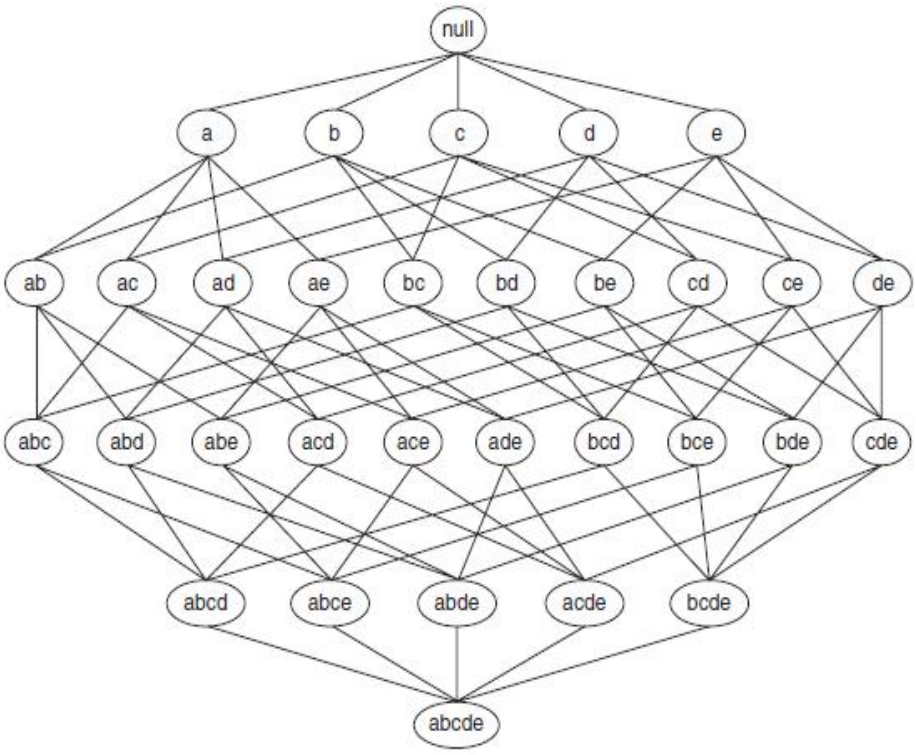


Figure 6.1 Item set lattice structure for the set I={a,b,c,d,e}

6.4.2 Apriori Principle

An item-set which has minimum support threshold value is called frequent item-set else it is called as infrequent item-set. Apriori principle is stated as “If an item set is frequent then all of its subsets must also be frequent”. It is used to reduce the number of candidate item-sets explored during frequent item set generation and this is called pruning.

To illustrate the idea behind the apriori principle, consider the item-set lattice shown in Figure 6.2. Suppose {*c, d, e*} is a frequent item-set. Clearly, any transaction that contains {*c, d, e*} must also contain its subsets, {*c, d*}, {*c, e*}, {*d, e*}, {*c*}, {*d*}, and {*e*}. As a result, if {*c, d, e*} is frequent, then all subsets of {*c, d, e*} must also be frequent. Conversely, if an item-set such as {*a, b*} is infrequent, then all of its supersets must be infrequent too.

The entire sub-graph containing the supersets of {*a, b*} namely {a, b, c}, {a, b, d}, {a, b, e}, {a, b, c, d}, {a, b, c, e},{a, b, d, e} and {a, b, c, d, e} can be pruned immediately once {*a, b*} is found to be infrequent as shown in Figure 6.2. This strategy of trimming the exponential search space based on the support measure is known as support-based pruning. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an item-set never exceeds the support for its subsets. This property is also known as the anti-monotone property of the support measure.

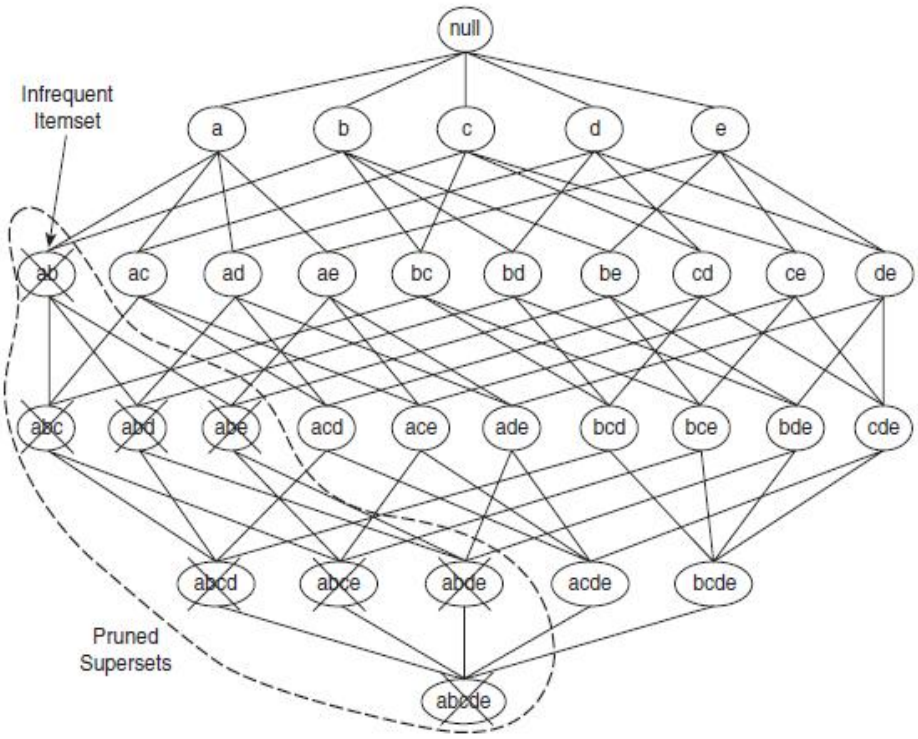


Figure 6.2 Apriori principle based pruning

6.4.3 Frequent item-set generation of the Apriori Algorithm

The pseudocode for the frequent item-set generation part of the Apriori algorithm is shown in Figure 6.3. Let C_k denote the set of candidate k-item-sets and L_k denote the set of frequent k-item-sets:

- The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-item-set will be known.
- Next, the algorithm will iteratively generate new candidate k-item-sets using the frequent (k−1) item-sets found in the previous iteration.

```
L1= {frequent items};
    for (k= 2; Lk-1 != φ ; k++) do begin
        Ck= candidates generated from Lk-1
      /* Self join  Lk-1 x Lk-1 and eliminating any k-1 size
         item-set that is not  frequent */
    for each transaction t in database do
            increment the count of all candidates in
            Ck that are contained in t
            Lk = candidates in Ck with min_sup
return (L1 ∪ L2 ∪ L3…..Lk);
```

Figure 6.3 Apriori Algorithm to generate frequent item-set

6.5 PAIR RECOMMENDATION BASED ON ASSOCIATION RULE MINING

In this research, we have proposed a new method for pair recommendation using association rule. The association rule provides two measures namely *confidence* which measures the degree of correlation among pairs and *support* which measures the significance of the correlation. Association rules which satisfy minimum support and confidence are called strong rules. Our objective is to find such strong association rules. The steps involved in finding strong association rules are as explained in Figure 6.4.

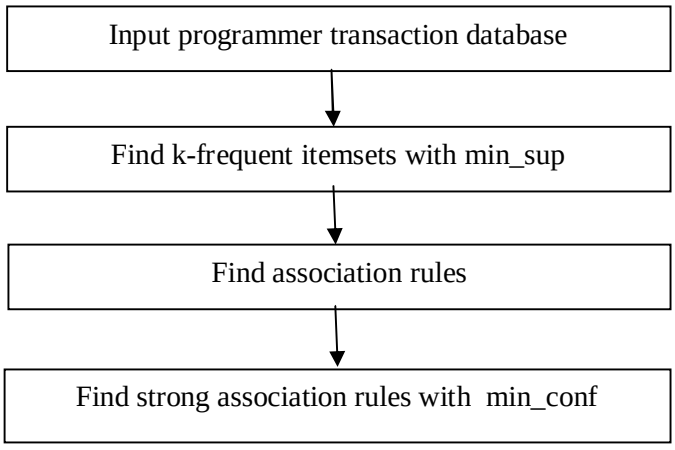


Figure 6.4 Finding of strong Association Rules

The detailed description of each of the above steps is explained in the following sub sections.

6.5.1 Programmer Transaction Database

The programmer transaction database contains list of transactions where each transaction is a record of programmers satisfying specific feature. An example transaction database containing list of programmers satisfying set of features is given in Table 6.2.

Let T = {A, B, C, D, E, F, G, H} are the programmers and their corresponding features are shown in Table 6.2

Table 6.2 An example Transaction database

Features	List of programmers
Phython	A, C, E, H
Java	A,B,C,D,E
Web Technology	D,E,H
CGPA >= 9	A,E,H
CGPA < 9	B,C,D
Music as hobby	B,D,E,H

For each feature, the list of programmers satisfying the feature is recorded. The feature Phython is known to programmers A, C, E and H and it is recorded as first transaction. The feature Java is known to programmers A, B, C, D and E. The feature Web Technology is known to programmers D, E and H. Programmers A, E and H has CGPA greater than or equal to 9. Programmers B, C and D have CGPA less than 9. In the last transaction, programmers B, D, E and H having music as their hobby is recorded.

6.5.2 Finding k-frequent item-set with minimum support

The frequent item-sets are obtained using Apriori algorithm. The key idea of the algorithm is to begin by generating frequent item-sets with just one item (i.e.) frequent 1-item-set and to recursively generate frequent 2-item-sets, then frequent 3-item-sets until frequent k-item-sets are generated. Minimum support count is a threshold value given by the user and it is used in finding the frequent k-item-sets.

6.5.3 Finding association rules

Association rules are generated after finding all frequent item- sets. For each frequent item-set L, all non empty subsets S is found and for each S, rule S⇒L - S is generated. For example if L⇒{A,B,C} is frequent, then its subsets are S={{A}, {B}, {C}, {A,B}, {A,C}, {B,C}, {A,B,C}}. Then the association rules are {A}⇒{B,C}, {B}⇒{A,C}, {C} ⇒{A,B}, {A,B}⇒{C}, {A,C}⇒{B} and {B,C}⇒{A}.

6.5.4 Finding strong association rules

Association rules satisfying minimum confidence level, which can be fixed by user, are called strong association rules. For example if minimum confidence threshold is 80% and the association rule {D} ⇒ {E.F} satisfies confidence measure 80%, then it is called strong association rule. This threshold value 80% is defined by the user depending on the application. This means that there is strong correlation between D and E, F. From the pair programming point of view D can be paired with programmer E as well as with programmer F.

6.6 EXAMPLE

Consider transaction database with 5 programmers P1, P2, P3, P4 and P5 with 9 features F1 to F9 as shown in Table 6.3.

Table 6.3 Programmer Transaction database

TID	List of Transactions
F1	P1,P2,P5
F2	P2,P4
F3	P2,P3
F4	P1,P2,P4
F5	P1,P3
F6	P2,P3
F7	P1,P3
F8	P1,P2,P3,P5
F9	P1,P2,P3

In the Table 6.3, TID refers to the transaction identifications namely F1, F2, F3, F4, F5, F6, F7, F8 and F9 which are features of programmers. For each transaction, we have corresponding list of programmers satisfying the features.

Let the minimum support be 2, which out of 9 transactions is 22 % and let minimum confidence required is 70%. First frequent item-set using Apriori algorithm is found. Then, Association rules will be generated using minimum support and minimum confidence.

The set of frequent 1-itemsets, L1, consists of the candidate 1-item-sets satisfying minimum support is shown in Table 6.4. For example, the item-set P1 occur 6 times in different transaction identifications namely, F1, F4, F5, F7, F8 and F9 and hence the support count of {P1} is taken as 6.

Table 6.4 Frequent 1-item-sets, L_1

Item-set	Support Count
{P1}	6
{P2}	7
{P3}	6
{P4}	2
{P5}	2

In the first iteration of the algorithm, each item is a member of the set of candidate. Next to discover the set of frequent k-item-sets, L_k, the algorithm uses self join L_{k-1} x $L_{k-1.}$ Initially, the set of frequent 2-item-sets, L_2, is found by self join L_1 x L_1 to generate a candidate set of 2-itemsets, C_2 which is shown in Table 6.5. From C_2, the item-sets which satisfies minimum support count is called L_2 and shown in Table 6.6. In Table 6.6, we can see that the support count value of item-set {P1,P2} is 4. In Table 6.3, the transactions {P1, P2} occurs 4 times, namely for the transaction identifications F1, F4, F8 and F9. Similarly the support count values for other item-sets in Table 6.6 are calculated.

Table 6.5 Candidate 2-item-sets, C_2

Item-set
{P1,P2}
{P1,P3}
{P1,P4}
{P1,P5}
{P2,P3}
{P2,P4}
{P2,P5}
{P3,P4}
{P3,P5}
{P4,P5}

Table 6.6 Frequent 2-item-sets, L_2

Item-set	Support Count
{P1,P2}	4
{P1,P3}	4
{P1,P5}	2
{P2,P3}	4
{P2,P4}	2
{P2,P5}	2

The generation of the set of candidate 3-item-sets, C_3, involves use of the Apriori Property. In order to find C_3, compute self join L_2 x L_2 and is shown in Table 6.7.

Table 6.7 Candidate 3-item-sets, C_3

Item-set
{P1,P2,P3}
{P1,P2,P5}
{P1,P3,P5}
{P2,P3,P4}
{P2,P3,P5}
{P2,P4,P5}

Now, Join step is complete and Prune step will be used to reduce the size of C_3. Prune step helps to avoid heavy computation due to large C_k. Based on the Apriori property that all subsets of a frequent item set must also be frequent, and hence the four latter candidates cannot possibly be frequent. For example, take {P1, P2, P3}. The 2-item subsets of it are {P1, P2}, {P1, P3} and {P2, P3}. Since all 2-item subsets of {P1, P2, P3} are members of L_2, {P1, P2, P3} is retained in C_3. Consider another example of {P2, P3, P5} which shows how the pruning is performed. The 2-item subsets are {P2, P3}, {P2, P5} and {P3,P5}. But {P3, P5} is not a member of L_2 and hence it is not

a frequent item-set. Thus {P2, P3, P5} is removed from C_3. Therefore, C_3= {{P1, P2, P3}, {P1, P2, P5}} after checking for all members of result of join operation for pruning. Now, the transactions in T are scanned in order to determine L_3, consisting of those candidates 3-itemsets in C_3 having minimum support and shown in Table 6.8.

Table 6.8 Frequent 3-item-sets, L_3

Item-set	Support Count
{P1,P2,P3}	2
{P1,P2,P5}	2

The algorithm uses self join L_3 x L_3 to generate a candidate 4-item-sets, C_4 and this is shown in Table 6.9.

Table 6.9 Candidate 4-item-sets, C_4

Item-set
{P1,P2,P3,P5}

C_4 contains only one item-set {P1, P2, P3, P5}and this is also pruned since its subset {P2, P3, P5}is not frequent. Thus, C4= φ, and algorithm terminates, having found all of the frequent item-sets. This completes Apriori Algorithm.

These frequent item-sets will be used to generate strong association rules where strong association rules satisfy both minimum support and minimum confidence. For each frequent item-set “L”, generate all nonempty subsets of L. For every nonempty subset S of L, output the rule “S ⇒ L - S” if supp(L) / supp(S) >= confidence, where confidence is minimum confidence threshold. The generation of strong association rule is shown in Figure 6.5.

Consider the frequent item set

L_k = {{P1}, {P2}, {P3}, {P4}, {P5}, {P1,P2}, {P1,P3}, {P1,P5}, {P2,P3}, {P2,P4}, {P2,P5}, {P1,P2,P3}, {P1,P2,P5}}.

Consider L = {P1,P2, P5}. All nonempty subsets of L are {P1,P2}, {P1,P5}, {P2,P5}, {P1}, {P2}, {P5}.

Let minimum confidence threshold is 70%. The resulting association rules are shown below, each listed with its confidence.

For rule R1: (P1 , P2) ⇒ P5

$$\text{Confidence} = \frac{\text{support count}(\{P1,P2,P5\})}{\text{support count}(\{P1,P2\})} = 2/4 = 50\%$$

Hence R1 is Rejected because it is less than the user defined threshold value which is 70%.

For rule R2: (P1, P5) ⇒ P2

$$\text{Confidence} = \frac{\text{support count}(\{P1,P2,P5\})}{\text{support count}(\{P1,P5\})} = 2/2 = 100\%$$

R2 is Selected, because it is higher the threshold value 70%.

For rule R3: (P2 , P5) ⇒ P1

$$\text{Confidence} = \frac{\text{support count}(\{P1,P2,P5\})}{\text{support count (P2, P5)}} = 2/2 = 100\%$$

R3 is Selected since it is higher than the threshold value.

For rule R4: P1 ⇒ (P2, P5)

$$\text{Confidence} = \frac{\text{support count}(\{P1,P2,P5\})}{\text{support count}(\{P1\})} = 2/6 = 33\%$$

R4 is Rejected, since it is less than the threshold value.

For rule R5: P2 ⇒ (P1, P5)

$$\text{Confidence} = \frac{\text{support count}(\{P1,P2,P5\})}{\text{support count}(\{P2\})} = 2/7 = 29\%$$

R5 is Rejected since it is less than the threshold value.

For rule R6: P5 ⇒(P1 , P2)

$$\text{Confidence} = \frac{\text{support count}(\{P1,P2,P5\})}{\text{support count}(\{P5\})} = 2/2 = 100\%$$

R6 is Selected since it is higher than the threshold value.

Figure 6.5 Generation of Strong Association Rule

Thus three strong association rules are verified.

6.7 EXPERIMENTAL RESULTS AND DISCUSSION

Experiments were conducted by choosing 40 students from fourth year of Five Year Integrated M.Sc Software Engineering and M.Sc Theoretical computer science. These students have to submit mini project/package for their elective courses. The resume of all 40 students were collected which contains data about their computer language proficiency, grades obtained, hobbies etc. The correlation among the students is found by applying Apriori algorithm.

Based on the correlation results obtained, 15 pairs and 10 solos were formed. Two packages were given related with the course Semantic Web. Package-1 was assigned to both pair programmers and solo programmers. The productivity measures, namely lines of code and time taken to complete the package-1 for pairs is recorded and shown in Figure 6.6 and 6.7 respectively.

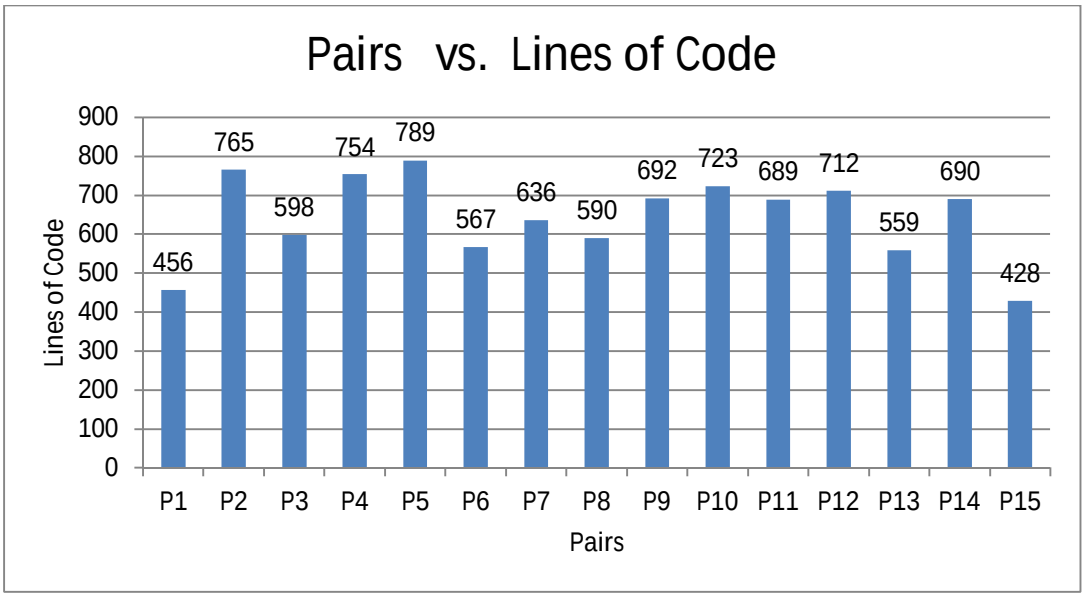


Figure 6.6 Lines of Code Analysis for the pairs

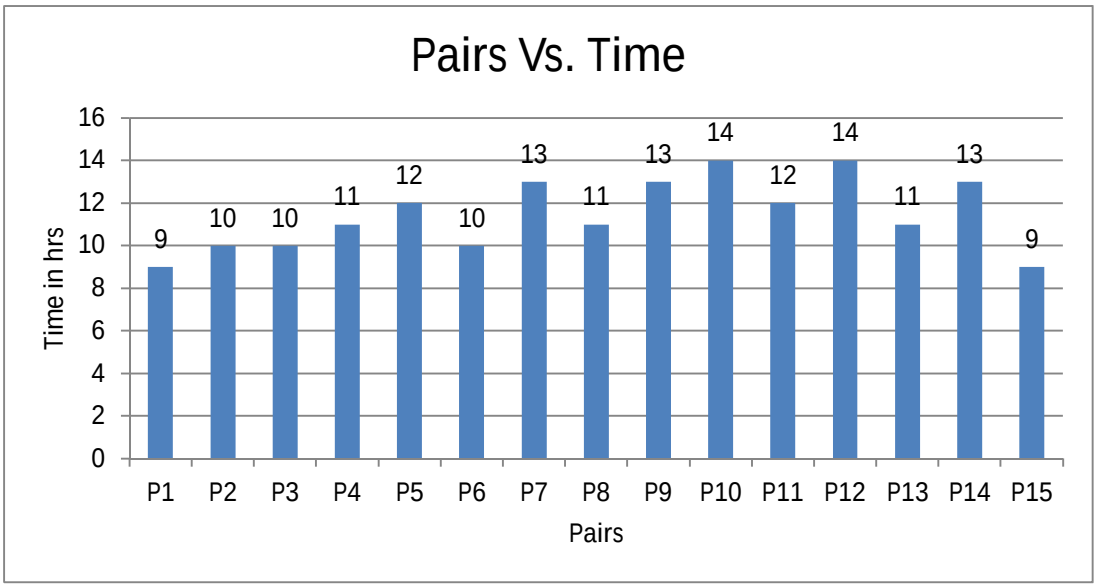


Figure 6.7 Time Analysis for the pairs

The pairs in the same group were changed for the package 2 in Semantic Web course and totally new 15 pairs were formed based on the recommended association rules. The lines of code and time taken for the new pairs are shown in Figure 6.8 and 6.9 respectively.

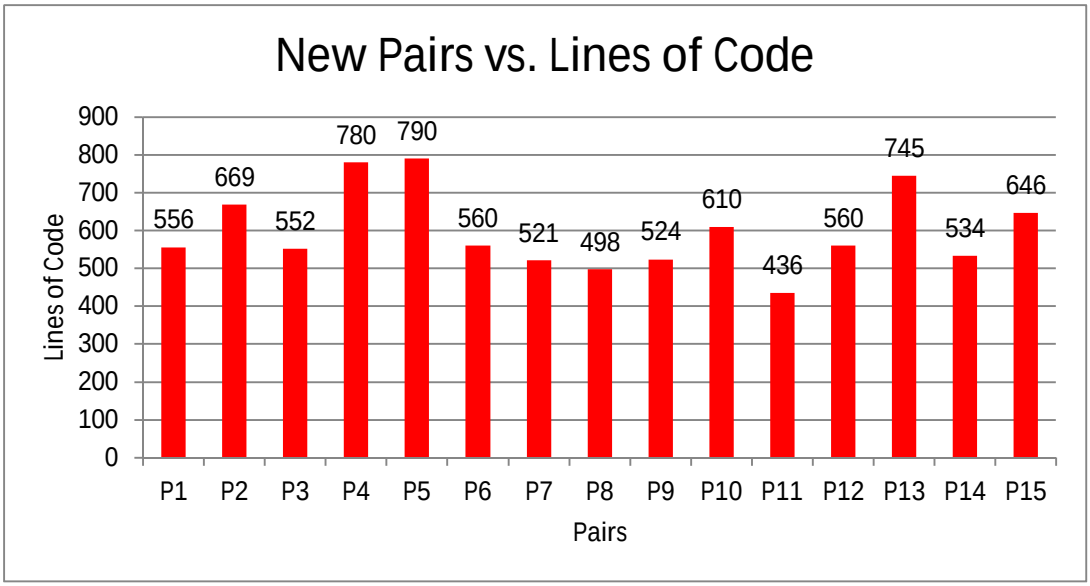


Figure 6.8 Lines of Code Analysis for the new pairs

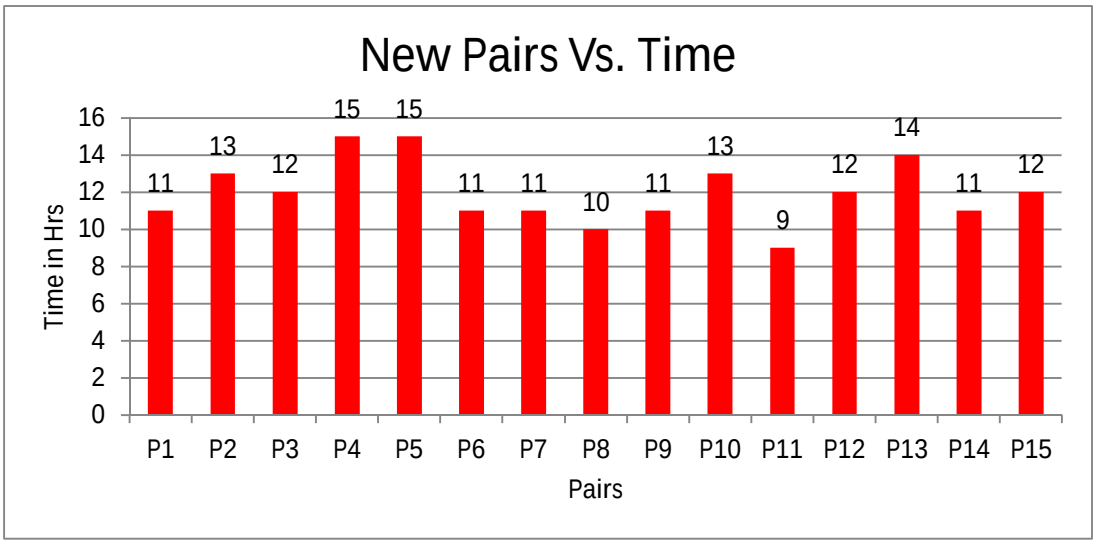


Figure 6.9 Time Analysis for the new Pairs

The average number lines of code and time taken to complete the package for the package-1 is 643 LOC and 10.8 hours respectively. After changing the pairs for the package-2, the average number lines of code and time taken to complete the package is 598 LOC and 12 hours respectively. This shows that the performance of students with respect to package-1 and with that of the package-2 is almost same. The productivity measures like lines of code and time taken to complete the package for solo programmers are shown in Figure 6.10 and 6.11 respectively.

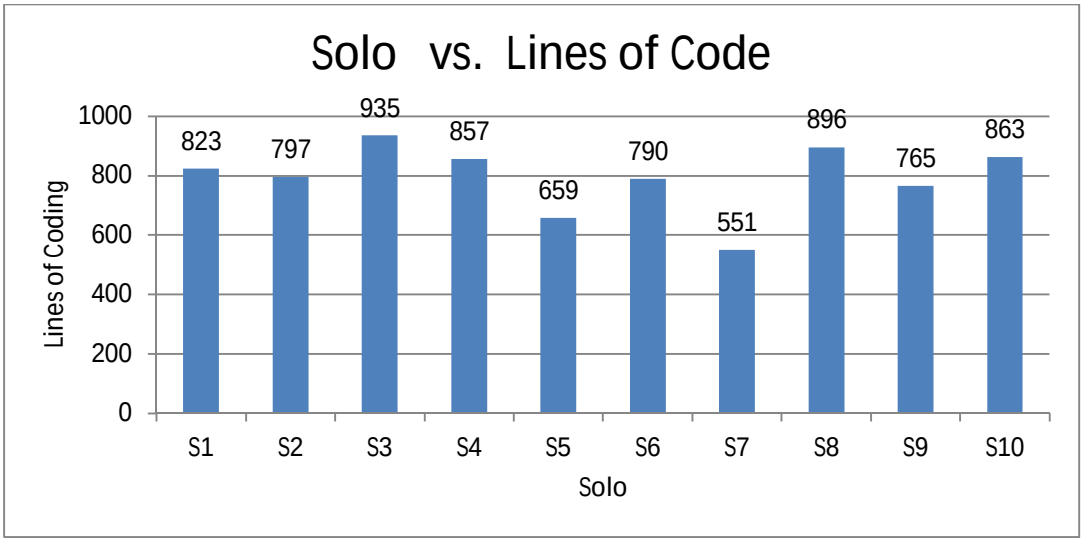


Figure 6.10 Lines of Code Analysis for solo programmers

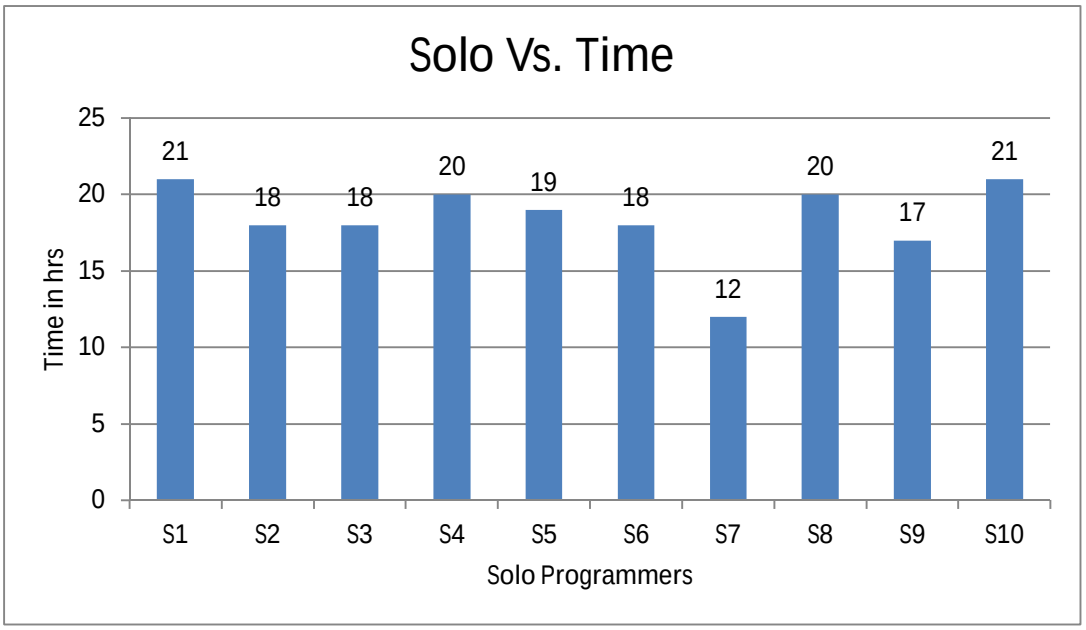


Figure 6.11 Time Analysis for solo programmers

The performance of pairs and new pairs is compared with solo programmers in terms of productivity measures time and LOC and the results are tabulated in Table 6.10 and 6.11 respectively.

Table 6.10 Performance of Solo Programmers Vs Pair

Productivity Measure	Pair	Solo	Percentage of Increase (Solo to Pair)
Time(in hrs)	10.8	18.4	70.37
LOC(Lines of code)	643	793	23.32

Table 6.11 Performance of Solo Programmers Vs New Pair

Productivity Measure	New Pair	Solo	Percentage of Increase (Solo to Pair)
Time(in hrs)	12	18.4	53.33
LOC(Lines of code)	598	793	32.60

From Table 6.10 and 6.11 we can observed that there is 23.32% increase in LOC and 70.37% in time when compared with pair and there is an increase of 32.60% in LOC and 53.33% in time when compared with new pairs.

Performance of students is measured by obtaining ratings from the faculty by using 10 point grade scale. The grade obtained by pair programmers for package-1 and package-2 and for solo programmers is shown in Figure 6.12, 6.13 and 6.14 respectively.

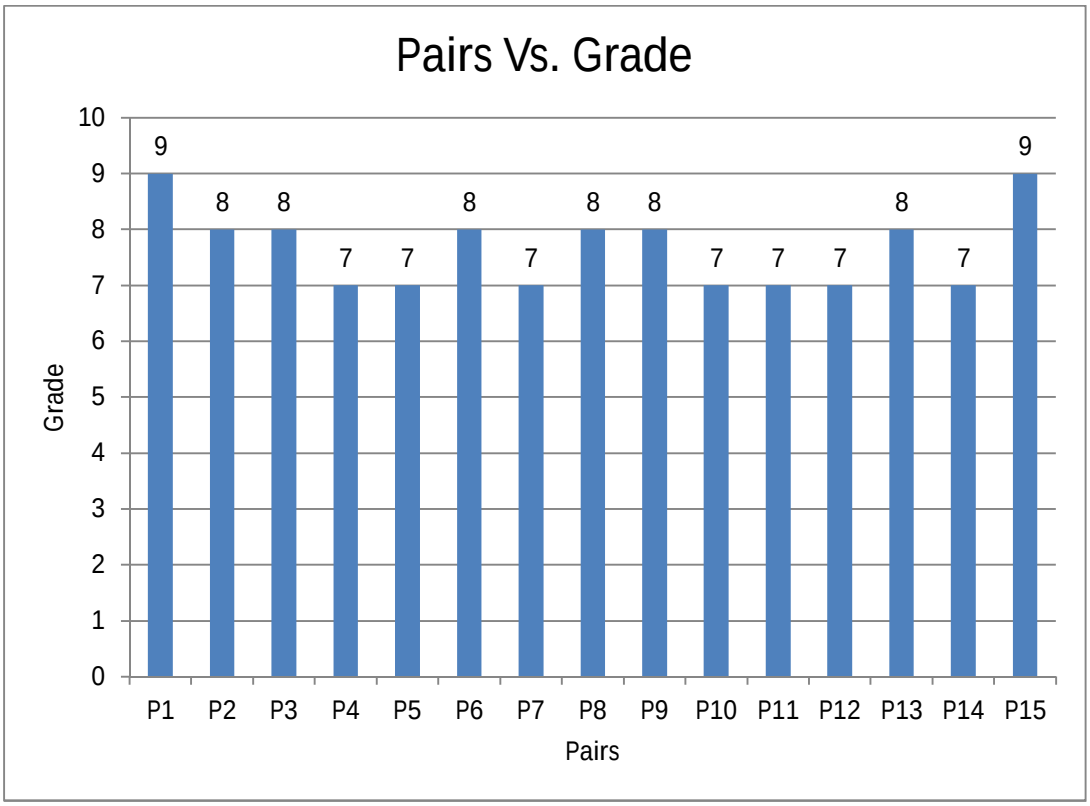


Figure 6.12 Grades obtained by pairs for package-1

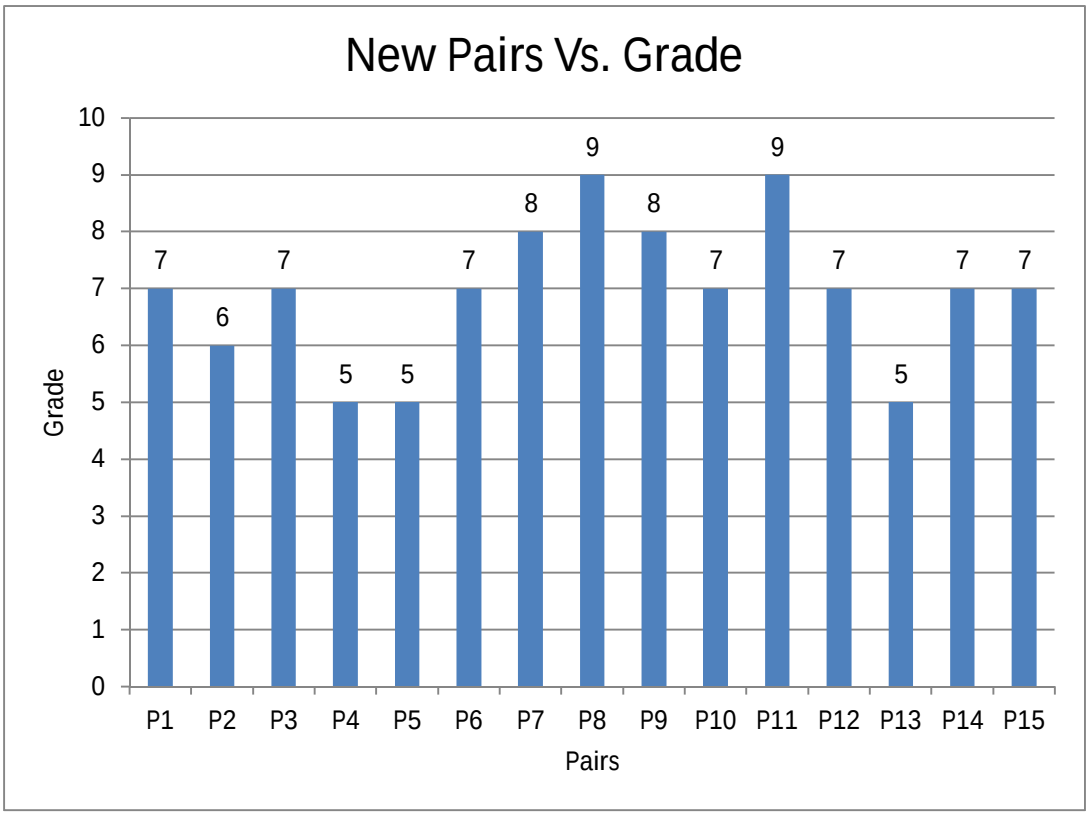


Figure 6.13 Grades obtained by new pairs for package-2

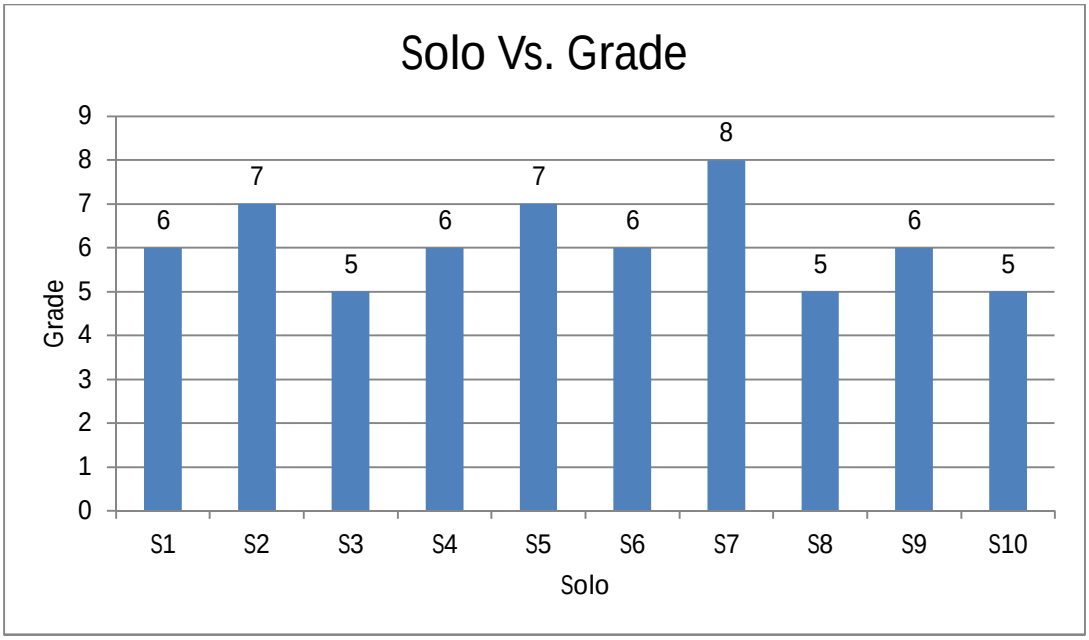


Figure 6.14 Grades obtained by solo programmers

From the above graphs it is observed that the average grade obtained by pairs for package-1, new pairs for package-2 and solo programmers is 7.6, 6.93 and 6.1 respectively. The grade obtained by solo programmers is much less than the grades obtained by pairs 1 and 2. This indicates pair programming improves the grades obtained by the students.

6.8 CONCLUSION

A new method for recommending pairs based on association rule mining is proposed. The productivity rate in terms of lines of code and time is measured for recommended pair 1 and pair 2 is found. The experimental result shows that there is no significant change in terms of productivity measure for pair 1 and pair 2. Also there is a small deviation in grades obtained by pair 1 and pair 2. The productivity rate of solo programmers is measured and this is less when compared with pair 1 and pair 2. Also the grades obtained by solo programmers are much less when compared with pair 1 and pair 2. This indicates that pair programming improves the knowledge and skill level of the students.

CHAPTER 7

CONCLUSION

This thesis analyses the objective and motivation of distributed pair programming which is one of the practices of Extreme Programming.

7.1 SUMMARY OF THE THESIS

Experiments have been conducted to evaluate the performance of students who were engaged in distributed pair programming during laboratory sessions with those who worked solo during the laboratory sessions. Even though the laboratory sessions did not add directly to the final grade, the outcome of being involved in a distributed pair programming experience appear to have enhanced the quality of self-governing assignment work. Furthermore, the majority of students enjoyed the practice and would like to have distributed pair programming used in future courses. The results provide the support for use distributed pair programming in the software engineering curriculum.

Pair programming which is a part of Agile software development method has been one of the leading research areas. Mostly such research setups are academic setup where both the programmers are in the pair co-located. This will not be case when we experiment in real time programmers in the industry. Hence the need for attempting distributed pair Programming arises. Our intension is to attack the problem of pair dismissal where either both or one of the pair trying to omit sharing of knowledge and lead the team

as a solo programmer. As a future work, we would provide a tool including usage of social networking platforms to avoid pair dismissal problem.

Pair programming is definitely one of the best mutual teaching-learning methodologies when the pairs are compatible and has the drive to achieve. The feedback survey conducted during our case study revealed that pairs of the third experiment were more comfortable and enthusiastic, which favors the significance of skill level in pairing. It was evident from the third experiment that the pairs were more compatible and they produced promising results. Essentially the success of pair programming depends on both the complexity of programming task and compatibility of the pairs. Only when the pairs are compatible with each other, the working environment will be more interesting for the pairs, which will improve their productivity. It will be a win-win strategy for both the partners where the job is expected to be completed successfully.

The study on using the pair programming for the programming laboratory courses in e-learning teaching-learning process in four laboratory courses offered at UG and PG level proved to be effective and useful. This approach has resulted in benefits such as enhancement of problem solving skills, efficiency, quality, trust, and teamwork skills. We have also observed that paired laboratory experience is especially advantageous to e-learners. A hidden advantage that was evident from the learners' responses was that learners were motivated to work collaboratively even for other tasks. Firstly, learners like this approach since they regard it as useful, facilitating the learning process, enabling to attain the learning goal and good learning experience.

Moreover, the results of this study indicate that the use of pair programming in e-learning has important effects on the learners' academic outcomes and also the dropout rate is also reduced. The learners were

motivated and involved in laboratory courses that created a confident in them. The learners obtained better final grades. Therefore, the results hereby presented suggest that this system can support effective learning strategies for laboratory courses in e-learning. The research showed several benefits of using pair programming in laboratory courses in e-learning such as enhanced learning, greater confidence in work quality, higher problem solving skills, enhanced interaction skills, and improved team building skills. The result also shows that e-learners had a good learning experience.

The majority of students enjoyed the practice and would like to have distributed pair programming used in future courses. The results provide the support for use distributed pair programming in the software engineering curriculum.

A new method for recommending pairs based on association rule mining was proposed. The productivity rate in terms of lines of code and time is measured for recommended pair 1 and pair 2 was found. The experimental result shows that there is no significant change in terms of productivity measure for pair 1 and pair 2. Also there is a small deviation in grades obtained by pair 1 and pair 2. The productivity rate of solo programmers is measured and this is less when compared with pair 1 and pair 2. Also the grades obtained by solo programmers are much less when compared with pair 1 and pair 2. This indicates that pair programming improves the knowledge and skill level of the students.

7.2 FUTURE WORK

The work carried out on pair programming using graph matching can be extended to distributed programming environment where the pairs will be geographically separated and the communication between the pairs can be through text chatting, voice or video conferencing. Moreover there is scope

for enhancing the measure of compatibility by encompassing many other related factors.

The study on pair programming approach for e-learning environment also indicated several areas for future research. Future studies can examine the effects of pair forming and automatic formation of pairs. Future studies can examine the use of pair programming in non computer science curriculum.

The results of the experiments indicate the following:

- Pair programming in virtual teams is a feasible way of developing object-oriented software.
- Pair programming in co-located teams is a feasible way of developing object-oriented software.
- Software development involving, distributed pair programming seems to be comparable to co-located software development in terms of the two metrics, namely productivity (in terms of Lines of Code per hour) and quality (in terms of the grades obtained).
- Co-located teams did not produce statistically significantly better results than the distributed teams.
- The feedback given by the students indicates that distributed pair programming fosters teamwork and communication within a virtual team.

The above results indicate that distributed pair programming can be employed as a very effective method in academic environment which will improve the productivity and quality of the student community.

REFERENCES

1. Abdullah Mohd ZIN, Sufian IDRIS & Nantha Kumar Subramaniam 2006, 'Improving Learning of Programming through E-Learning by Using Asychronous Virtual Pair Programming', Turkish Online Journal of Distance Education-TOJDE, vol. 7, no. 3, article-13.

2. Alistair Cockburn & Laurie Williams 2001, 'The Costs and Benefits of Pair Programming', Addison-Wesley, Massachusetts.

3. Alistair Cockburn, 'The Costs and Benefits of Pair Programming', Internet source. Beck, K 2000, Extreme Programming Explained: Embrace Change, Reading, Addison-Wesley, Massachusetts.

4. Bella, E , Fronza, I , Phaphoom, N , Sillitti, A , Succi, G & Vlasenko, J 2013, 'Pair Programming and Software Defects — A Large, Industrial Case Study', IEEE Trans. Software Engineering, pp. 930-953.

5. Bevan, J, Werner, L & McDowell, C 2002, 'Guidelines for the use of pair programming in a freshman programming class', Conference on Software Engineering Education and Training, IEEE Computer Society, pp. 100-107.

6. Braught, G, Wahls, T & Eby, L.M. 2011, 'The case for pair programming in the computer science classroom', ACM Transactions on Computing Education.

7. Bravo, C, Duque, R , Gallardo, J, García, J & García, P 2007, 'A Groupware System for Distributed Collaborative Programming: Usability Issues and Lessons Learned', International Workshop on Tools Support and Requirements Management for Globally Distributed Software Development, Centre for Telematics and Information Technology, pp. 50–56.

8. Brian Hanks, F 2004, 'Distributed Pair Programming: An Empirical Study', Extreme Programming and Agile Methods - XP/Agile Universe, Proceedings.

9. Canfora, Cimitile, GA & Visaggio, CA 2003, 'Lessons learned about distributed Pair programming: what are the knowledge needs to address?', Proceedings Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 314 – 319.

10. Carver, JC, Henderson, L, He, L, Hodges, JE & Reese, DS 2007, 'Increased retention of early computer science and software engineering students using pair programming', Conference on Software Engineering Education and Training, pp. 115-122.

11. Chaparro, EA, Yuksel, A, Romero, P & Bryant, S 2005, 'Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education', 17th Workshop of the Psychology of Programming Interest Group, Sussex University, pp. 5–18.

12. Cliburn, D.C 2003, 'Experiences with pair programming at a small College', Journal of Computing Sciences in Colleges, USA, vol. 19, no.1, pp. 20-29.

13. Cockburn, A & Williams, L 2000, 'The cost and benefits of pair programming', eXtreme Programming and Flexible Processes in Software Engineering (XP2000), pp. 223-247.

14. Concas, G 2007, '(Eds.): XP 2007', LNCS 4536, pp. 70–73.

15. David Stotts, Laurie Williams, Nachiappan Nagappan, Preshant Baheti, Dennis Jen & Anne Jackson 2003, 'Virtual teaming: Experiments and experiences with distributed pair programming', In Extreme Programming and Agile Methods – XP/Agile Universe, Springer, no. 2753, pp. 129-141.

16. DeClue, TH 2003, 'Pair programming and pair trading: effects on learning and motivation in a CS2 course', Journal of Computing Sciences in Colleges, USA, vol.18, no.5.

17. Denner, J, Werner,L, Campe, S & Ortiz, E 2014, 'Pair Programming: Under What Conditions Is It Advantageous for Middle School Students?', Journal of Research on Technology in Education, vol.46, no.3, pp. 277-296.

18. Donohue, T. L & Wong, E. H 1997, 'Achievement motivation and college satisfaction in traditional and nontraditional students. Education', vol. 118, no.2, pp. 237– 244.

19. Dybå, T;Arisholm, E;Sjøberg, D;Hannay, J & Shull, F 2007, 'Are Two HeadsBetter Than One? On the Effectiveness of Pair Programming', IEEE Software, vol. 24, no. 6, pp. 12-15.

20. Engelbart, DC & English, WK 1968, 'A Research Center for Augmenting Human Intellect', presented at AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference, San Francisco, CA.

21. Erdogmus, H & Williams, L 2003, 'The Economics of Software Development by Pair Programmers', The Engineering Economist, vol. 48, no. 4, pp. 283-319.

22. Frank Maurer 2002, 'Supporting distributed extreme programming', In Extreme Programming and Agile Methods - XP/Agile Universe *2002*, no. 2418 in LNCS, pp. 13-22.

23. Furberg, A, Kluge, A & Ludvigsen, S 2013, 'Student sense making with science diagrams in a computer-based setting', International Journal of Computer-Supported Collaborative Learning, vol.8, no.1, pp. 41-64.

24. Gehringer, EF 2003, 'A pair-programming experiment in a non-programming course', Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.

25. George, B & Mansour, YM 2002, 'A Multidisciplinary Virtual Team', Accepted at Systemics, Cybernetics and Informatics (SCI).

26. Gerardo Canfora, Aniello Cimitile & Corrado Aaron Visaggio 2003, 'Lessons learned about distributed pair programming: What are the knowledge needs to address?', In Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE03), pp. 314-319.

27. Gittins, R & Hope, S 2001, 'A study of Human Solutions in eXtreme Programming', 13th Workshop of the Psychology of Programming Interest Group, pp. 41-51.

28. Hahn, Jan Hendrik, Mentz, Elsa Meyar & Lukas 2009, 'Assessment Strategies for Pair Programming', Journal of Information Technology Education, vol. 8, pp. 273-284.

29. Hanks, B 2003,'Empirical Studies of Pair Programming', Proceedings of 2nd International Workshop on Empirical Evaluation of Agile Processes (EEAP *'03)*.

30. Hanks, B, McDowell, C, Draper, D & Krnjajic, M 2004, 'Program Quality with pair programming in *CS1'*, Proceedings of 9th Annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '04), pp.176-180.

31. Hiroshi Natsu 2003, 'Distributed Pair Programming on the Web', Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'03).

32. Jason, A 2004, 'Technical and human perspectives on pair programming', ACM SIGSOFT Software Engineering Notes, vol. 25, no.5, pp.1-14.

33. Jim Highsmith 2001, 'History : The Agile Manifesto', www.agilemanifesto.org.

34. Katira, N, Williams, L, Wiebe, E, Miller, C, Balik, S & Gehringer, E 2004, 'Paired programming/ collaborative learning: On understanding compatibility of student pair programmers', Proceedings of the 35th SIGCSE technical symposium on Computer science education, pp. 7-11.

35. Layman, L 2006, 'Changing students perceptions: An analysis of the supplementary benefits of collaborative software development', Proceedings of 19th Conference on Software Engineering Education and Training, pp. 159-166.

36. Levy, Y 2007, 'Comparing dropouts and persistence in e-learning courses', Computers & Education, vol.48, no.2, pp. 185–204.

37. Maguire, P & Maguire, R 2013, 'Can Clickers Enhance Team Based Learning? Findings from a Computer Science Module', AISHE-J: The All Ireland Journal of Teaching & Learning in Higher Education, vol.5. no.3.

38. McDowell, C, Hanks, B, Werner, L 2003, 'Experimenting with pair programming in the classroom', SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE), pp.60-64.

39. McDowell, L, Werner, H.E, Bullock, J & Fernald, J 2006, 'Pair programming improves student retention, confidence and program quality', Communications of the ACM, vol.49, no.8, pp.90-95.

40. McDowell, C, Hanks, B & Werner, L 2003, 'Experimenting with pair programming in the classroom', Proceedings of the 8th annual conference on Innovation and technology in computer science education, ACM SIGCSE Bulletin, vol.35, no.3.

41. McDowell, C, Werner, L, Bullock, EH & Fernald, J 2003, 'The impact of pair-programming on student performance, perception and persistence', Proceedings of the 25th International Conference of Software Engineering, Oregon, pp. 602-607.

42. McDowell, C, Werner, L, Bullock, H & Fernald, J 2002, 'The effects of pair-programming on performance in an introductory programming course', Proceedings of the 33rd SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, vol.34, no.1.

43. Nagappan, N, Williams, L, Ferzli, M, Wiebe, E, Yang, K, Miller, C & Balik, S 2003,'Improving the CS1 experience with pair programming', Proceedings of the 34th ACM SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, vol.35, no.1.

44. Natsu, H, Favela, J, Moran, AL, Decouchant, D & Martinez-Enriquez, AM 2003, 'Distributed pair programming on the Web', Proceedings of the Fourth Mexican International Conference on Computer Science, pp. 81–88.

45. Oncu, S & Cakir, H 2011, 'Research in online learning environments: priorities and methodologies', Computers & Education, vol.57, no.1, pp.1098–1108.

46. Padberg, F & Müller, M 2003, 'Analyzing the Cost and Benefit of Pair Programming', International Software Metrics Symposium (METRICS), Sydney, Australia, pp. 166 – 177.

47. Prashant Baheti, 'Exploring Pair Programming in Distributed Object-Oriented Team Projects'.

48. Prashant Baheti, Edward Gehringer & David Stotts 2002, 'Exploring the efficacy of distributed pair programming', In Extreme Programming and Agile Methods - XP/Agile Universe 2002, no.2418 in LNCS, pp. 208-220.

49. Prashant Baheti, Laurie Williams & Edward Gehringer, 'Distributed Pair Programming: Empirical Studies and Supporting Environments', Report of Department of Computer Science North Carolina State University Raleigh, NC 27695.

50. Preston, D 2005, 'Pair programming as a model of collaborative learning: A review of the research', Consortium for Computing Sciences in Colleges, pp.39-45.

51. Rakesh Agarwal, Tomasz Imielinski & Arun Swami 1993, 'Mining Association Rules between Sets of Items in Large Databases', Proceedings of the 1993 ACM SIGMOD conference, Washington D.C.

52. Ron Jeffries, Ann Anderson & Chet Hendrickson 2000, Extreme Programming Installed, Addison-Wesley.ISBN 0-201-70842-6.

53. Rosenberg, MJ 2001, 'E-learning: strategies for delivering knowledge in the digital age', McGraw Hill, New York,

54. Salleh, N, Mendes, E & Grundy, J 2011, 'Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review, IEEE Transactions on Software Engineering, vol.37 no.4, pp. 509-525.

55. Sillitti, A, Succi, G & Vlasenko, J 2012, 'Understanding the Impact of Pair Programming on Developers Attention - A Case Study on a Large Industrial Experimentation', 34th IEEE International Conference on Software Engineering (ICSE 2012), pp. 1094-1101.

56. Srikanth, H, Williams, L, Wiebe, E, Miller, C & Balik, S 2004, 'On pair rotation in the computer science course', Proceedings of 17th Conference on Software Engineering Education and Training, pp. 144 – 149.

57. Straub, DW 1989, 'Validating instruments in MIS research', MIS Quarterly, vol.13, no.2, pp.147–169.

58. Sultan Alshehri & Luigi Benedicenti, 2014, 'Ranking and Rules for selecting two persons in Pair Programming', Journal of Software, vol. 9, pp. 2467-2473.

59. Tessem, B 2003, 'Experiences in Learning XP Practices: A Qualitative Study', 4th International Conference on Extreme Programming and Agile Processes in Software Engineering, pp. 131-137.

60. Thomas, L, Ratcliffe, M & Robertson, A 2003, 'Code warriors and code-a-phobes: study in attitude and pair programming', Proceedings of the 34th SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, vol.35, no.1.

61. Till Schummer & Jan Schummer 2001, 'Support for distributed teams in extreme programming', Giancarlo Succi and Michele Marchesi, editors, Extreme Programming Examined, Addison-Wesley, Massachusetts, pp.355-378.

62. Tomayko, J 2002, 'A Comparison of pair programming to Inspection for software Defect Reduction', Computer Science Education, vol.12, no.3, pp 213-223.

63. Tristan Richardson, Quentin Stafford-Fraser, Kenneth, R, Wood & Andy Hopper 1998, 'Virtual network computing', *IEEE* Internet Computing, vol.2, no.1, pp.33-38.

64. Tsai, WT, Li, W, Elston, J & Chen, Y 2011, 'Collaborative learning using wiki web sites for computer science undergraduate education: A case study', IEEE Transactions , vol.54, no.1, pp.114-124.

65. Van Der Vyver, G & Lane, M 2003, 'Using a Team-based Approach in an IS Course: An Empirical Study', Journal of Information Technology Education, pp.393-406.

66. VanDeGrift T 2004, 'Coupling pair programming and writing: Learning about students' perceptions and processes', Proceedings of the 35th SIGCSE technical symposium on Computer science education, pp. 2-6.

67. Vanhanen, J & Lassenius, C 2007, 'Perceived Effects of Pair Programming in an Industrial Context', 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, Lubeck, pp. 211 – 218.

68. Wells, D & Williams, L 2002, '(Eds.): XP/Agile Universe', LNCS 2418, pp. 13–22.

69. West, DB 2011,' Introduction to Graph Theory', Prentice Hall.

70. Williams, L & Upchurch, R 2001, 'In support of student pair programming', SIGCSE Conference on Computer Science Education, pp.327-331.

71. Williams, L & Kessler, R 2002,'Pair Programming Illuminated', Addison-Wesley.

72. Williams, L 2001, 'Integrating pair programming into a software development process', Proceedings 14th Conference on software Engineering Education and Training, pp. 27 – 36.

73. Williams, L, Kessler, R, Cunningham, W & Jeffries, R 2000, 'Strengthening the Case for Pair-Programming', IEEE Software, vol. 17, no. 4, pp. 19-25.

74. Williams, L, Kessler, RR, Cunningham, W & Jeffries, R 2000, 'Strengthening the case for pair programming', IEEE Software, vol. 17, no.4.

75. Williams, L, McDowell, C, Nagappan, N, Fernald, J & Werner, J 2003, 'Building pair programming knowledge through a family of experiments', Proceedings International Symposium Empirical Software Engineering, pp. 143 – 152.

LIST OF PUBLICATIONS

1. **Mohanraj, N,** Sankar, A 2011, 'Assessing the effectiveness of Distributed Pair Programming for an lab assignment in Software Engineering Curriculum', International Conference on Mathematical and Computational Models, Volume : ISBN 978-81-8487-164-7 , pp. 406-414.

2. **Mohanraj, N,** Sankar, A 2014, 'Distributed Pair programming : A Survey', International Journal of Engineering Research & Technology, vol. 3 no. 8, e-ISSN: 2278-0181, pp. 1-7.

3. **Mohanraj, N,** Lekshmi, R, S, Sankar, A, Vidhya, R 2014, 'Pair Programming : A Novel weighted Graph Matching Approach for Pair Compatibility', Australian Journal of Basic and Applied Sciences, vol. 8, ISSN : 1991-8178, pp. 384-393.

4. **Mohanraj, N,** Sankar, A, Senthil Kumaran, V 2015, 'Enhancing learning experience of e-learners in laboratory courses using pair programming', ARPN Journal of Engineering and Applied Sciences, vol. 10, no.8, ISSN : 1819-6608, pp. 3836- 3843.