Journal of Software Engineering and Applications, 2014, 7, 1008-1018
Published Online November 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.712088

An Empirical Study of the Optimum Team Size Requirement in a Collaborative Computer Programming/Learning Environment

Olalekan S. Akinola[*], **Babatunde I. Ayinla**

Department of Computer Science, University of Ibadan, Ibadan, Nigeria
Email: [*]Akinola.olalekan@dlc.ui.edu.ng, i.ayinla@mail.ui.edu.ng

Received 7 September 2014; revised 5 October 2014; accepted 1 November 2014

Abstract

Pair programming has been widely acclaimed the best way to go in computer programming. Recently, collaboration involving more subjects has been shown to produce better results in programming environments. However, the optimum group size needed for the collaboration has not been adequately addressed. This paper seeks to inculcate and acquaint the students involved in the study with the spirit of team work in software projects and to empirically determine the effective (optimum) team size that may be desirable in programming/learning real life environments. Two different experiments were organized and conducted. Parameters for determining the optimal team size were formulated. Volunteered participants of different genders were randomly grouped into five parallel teams of different sizes ranging from 1 to 5 in the first experiment. Each team size was replicated six times. The second experiment involved teams of same gender compositions (males or females) in different sizes. The times (efforts) for problem analysis and coding as well as compile-time errors (bugs) were recorded for each team size. The effectiveness was finally analyzed for the teams. The study shows that collaboration is highly beneficial to new learners of computer programming. They easily grasp the programming concepts when the learning is done in the company of others. The study also demonstrates that the optimum team size that may be adopted in a collaborative learning of computer programming is four.

Keywords

Optimum Team Size, Collaborative Learning, Collaborative Programming, Computer Programming

[*]Corresponding author.

1. Introduction

Computer Programming is known to be a complex task that is difficulty to do. The first and most challenging task encountered by computing students is how to understand computer programming. The difficulties faced by the students are attributed to the high degrees of failure and the difficulties presented in the courses directly dependent on the abilities to program, to develop a logical reasoning and to solve problems. According to Eustáquio [1], this is the result of the difficulties found by instructors to effectively guide the students during their programming lab activities, due to the large number of students per class.

Production of high-quality software has been assisted by pair or collaborative programming, which was formalized as one of the core practice in extreme programming (XP). The existing studies indicate some improved outcomes through collaborative programming, such as better quality software, faster production speed, fewer defects and greater enjoymet [2].

The history of pair programming stretches back to punched cards; it emerged as a viable approach to software development in the early 1990s when it was noted as one of the 12 key practices promoted by extreme programming (XP). According to Müller [3] in the work of Rajendran and David [4], industry and academia have turned their attention and interest toward pair programming in recent years. It has been widely accepted as an alternative to traditional individual programming.

Recent observations show that in collaborative learning environments, students are able to reach improvements on performance, critical thinking and cooperative behaviour. Collaborative work is based on the assumption that two or more individuals working together can reach a state of equilibrium where ideas can be exchanged and distributed among group members, generating as a result new ideas and knowledge [1].

However, many factors may affect the efficiency of collaborative learning. Dillenbourg [5] listed group composition, group size and individual differences between group members as conditions that could affect collaboration. He further asserts that having the right core team can make or break a project and that great care should be taken when selecting team members. World Health Organization (WHO) [6] opines that it is very useful to consider team size, overall team composition, team member selection and exclusion criteria for a collaborative work. LeJeune [7] argued in favour of small-groups interactions. She suggested that group size should be between five and seven people and that grouping more than seven people might result in communication problems.

Hence, one of the most difficult decisions in programming teamwork or collaboration task in computer science is how to determine how many people should be in a team. The current research in this area has not being precise on the appropriate team (group) size. This study seeks to empirically determine the optimum team size needed in a collaborative computer programming/learning environment.

2. Related Works

Current trend shows that within the next ten years, collaboration in software engineering will change in a number of ways and research will need to shift its focus to enable and enhance such collaboration. Specifically, Thomas and Christian [8] claim that software in the small will become more popular and even large software will be built by fewer people due to better tools. For large projects, research will need to address the collaboration needs of project members other than just developers, including quality assurance engineers, build engineers, architects, and operations managers. According to Edgar *et al.* [9], collaborative programming improves the quality of software design, reduces the deficiencies of the code, enhances technical skills, improves team communication and it is considered to be more enjoyable for the participants.

In pair programming, two programmers work collaboratively at one computer on the same design, algorithm, or code. Prior research on this style of programming has primarily focused on its evaluation in academic settings [10]. This practice has been nominated several times in the last decades as an improved way of developing software [11]. In pair programming, the two programmers are like a unified, intelligent organism working with one mind, responsible for every aspect of this artifact. One partner, the driver, controls the pencil, mouse, or keyboard and writes the code. The other partner continuously and actively observes the driver's work, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications. The partners deliberately switch roles periodically. Both are equal, active participants in the process at all times and wholly share the ownership of the work product, whether it is a morning's effort or an entire project [12].

Recently, academic attentions are shifting to the practice of collaboration, as more and more commercial companies consider its use. Observations show that most professional programmers do not work alone, but rather on a software development team especially as the complexity and size of modern software projects rise. It then shows that with the increasing need to coordinate work; programming work has had more and more of a social component. Programmers commonly turn to team members for technical knowledge, advice and programming help.

It would be an understatement to say that deciding on the size of a team is an exact science! It is certainly a topic where there are a lot of controversial views. At a basic level, large teams have always been criticized of being unwieldy and ineffective in delivering results while small teams are perceived to be a lot more sure footed and better at delivering results. Despite these generalizations, most experts say that the ideal team size lies somewhere between 5 - 10 individuals [13]. Over the years, there has been a lot of curiosity and research on this issue and there are several interesting findings pertaining to various business situations. It is asserted that having the right core team can make or break a project and that is why great care needs be taken when selecting team members [6].

Usually collaborative programming teams are usually of two types. The egoless or decentralized programming team normally contains groups of ten or fewer programmers. The group members set the goals and codes are exchanged among them. Leadership is rotated within the group according to the needs and abilities required during a specific time. However, the egoless team is criticized of lack of structure which usually results in lower efficiency, effectiveness and error detection for large-scale projects. Egoless programming teams work best for tasks that are very complex. However, individuals that had decentralized programming team experience report higher job satisfaction [14].

The chief programmer (centralized) team usually contains three-person teams, consisting of a chief programmer, senior level programmer and a program librarian. Additional programmers and analysts are added to the team when necessary. The weaknesses of this structure include a lack of communication across team members, task cooperation and complex task completion. The centralized team works best for tasks that are more simple and straightforward since the flow of information in the team is limited. Individuals that had experience in this team structure typically report lower work morale [14].

In this study, we explore the effectiveness of the egoless collaborative team structure by organizing some experiments to determine the optimum team size needed in a typical programming environment.

3. Research Methodology

3.1. Subjects

The subjects used for this research were the 90 volunteered Students in 200 level of the Department of Computer Science, University of Ibadan, Nigeria. These Students were chosen as subjects for this project because it is appropriate to use students who are just learning how to program because the effect of collaboration can easily be noticed without wasting much resource. The experiments were carried out while they were learning Java Computer Programming in the first semester of 2012/2013 session for thirteen (13) weeks.

3.2. Experimental Design/Setting

Two experiments were organized in the study. The first involved team compositions of different genders. The students were randomly grouped into five parallel teams of different sizes ranging from 1 to 5. Each team size was replicated six times. That is, within a team, we have 6 groups working independently and collaboratively on the same problem. The teams consist of mixtures of both male and female students but not necessarily of equal numbers. The second experiment involved teams of same gender compositions (males or females). Only one set of parallel teams was involved in the second experiment. The average of the times and performances were determined and recorded for each team size.

The designed instruments for this experiment were two practical questions that involved the use of object oriented programming concepts, databases, and data structures as well as control structures. Each parallel teams were served with Report Forms that were filled during each stages of programming. During the experiment each team recorded the start time and the end time for the analysis, coding, compilation and debugging of the given problem. Number of errors (bugs) incurred during the code compilation were also recorded by the different teams.

3.3. Conducting the Experiment

The first experiment was carried out on the 8th of March, 2013 around 10 a.m. which lasted for five hours. This experiment involved random team sizes of different gender compositions. The second experiment involving single gender teams (males and females) of sizes 1 to 5 was thereafter conducted two weeks after the first experiment. The experiment was monitored and conducted by the researchers. The researchers served as the coordinators for the periods of the experiment. The team identification number was filled on the Reporting forms in order to identify each team. Analysis of the individual Reporting forms to elicit the data for identification of time spent on coding, compiling and debugging reported by recording officer were done thereafter. The efforts (time taken for analysis, coding and debugging) of each team during the experiment was analyzed.

The teams were given same programming problem to solve right from the analysis stage through compilation and debugging to final execution in order to avoid biasing of the results obtainable from the experiments. Moreover, the researchers ensured that there was no interaction between different groups/teams during the experiments such that ideas were not shared via groups. During the sessions, one member of the team served as recording officer, one served as typist and another served as coordinator. All errors (bugs) and times were recorded by the recording officers on the Reporting forms.

3.4. Variables

The experiment manipulated dependent and independent variables. The independent variables manipulated were the team sizes (1 to 5) and gender teams (male and female). The dependent variables measured were:

1) Analysis Effort: Analysis time is the time spent by a team in understanding the problem. The analyses done by the teams were based on their:

- Explanation of the understanding of the problem;
- Identification and Design of inputs and outputs variables, their data types and their sources (keyboard, database or file) or destination (monitor, file or database);
- Use of appropriate programming control structures (selection and loop controls);
- Design of output interface;
- Use of some concepts of object oriented programming-methods, classes and objects.

2) Coding Effort: Coding time is the time spent by a team to translate the analysis in to a chosen computer programming language.

3) Compilation Effort: Compilation time is the time spent in translating the code to machine language.

4) Debugging Effort: Debugging time is the time spent to debug programs by a team.

5) Number of Bugs: Number of bugs is the numbers of errors recorded during compilation.

6) Team's Program Accuracy: Adherence to problem specification, *i.e.*, solving all the problem's specifications completely. This was scored over 10 marks.

7) Team's Program Efficiency: Efficiency is determined by the total lines of code generated, use of proper programming logic and control structures and use of object oriented concepts. This was also scored over 10 marks.

8) Team's Effectiveness: The addition of the scores obtained in Accuracy and Efficiency.

9) Best Effective Team: The team with least analysis and programming effort (coding time), best effectiveness and having the least number of bugs incurred.

3.5. Threats to Validity

The question of validity of an experiment draws attention to how far a measure really measures the concept that it purports to measure [15]. Therefore in this experiment, we considered two important treats that may affect the validity of the outcomes of this study.

3.5.1. Threats to Internal Validity

Threats to internal validity are influences that can affect the dependent variable without the researcher's knowledge [16]. We considered three such influences: 1) Selection effects; 2) Maturation effects; 3) Instrumentation effects.

Selection effects are due to natural variations in human performance. For example, if the experiment is done only by highly experienced people, their average skill can be mistaken for a difference in the effectiveness of the treatments. We limited this effect by randomly assigning the students for each programming task, this way individual difference was spread across all treatments. Maturation effects result from the participants' skills improving with experience. Randomly assigning the students and doing the experiments within the same period of time checked this effect. Each student had the chance of being in one team for once. Instrumentation effects are caused by the problem given to students to solve, by difference in data collection forms, or by other experimental materials. In this study, this was very negligible or of time. Again, one set of data collection forms was used for the entire team and the same problems were solved by the teams.

3.5.2. Threats to External Validity

Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice [16]. We considered three sources of such threats: 1) experimental scale; 2) subject generalizability; 3) subject and artifacts representativeness.

Experimental scale is a threat when the experimental setting or the materials are not representative of industrial practice. This experiment was carried out in an academic setting and so this limit the extension of the results obtained to industrial setting. A threat to subject generalizability may exist when the subject population is not drawn from the industrial population, while threats regarding subject and artifact representativeness arise when the subjects and programming problems are not representatives of the industrial population. Same issue is attributed to these threats.

4. Results and Discussion

4.1. Experiment 1: Randomly Assigned Teams with Gender Mixture Compositions

Table 1 and **Figure 1** give the overall average mean performances for the six parallel groups having five different team sizes each.

Table 1. Overall average teams' performance.

Team Size	Mean Analysis Time (Minutes)	Mean Coding Time (Minutes)	Mean Effectiveness	Mean Compile-Time Error (Bugs) Incurred
1	36.3	81.2	19.5	21
2	44.3	80.0	16.5	11
3	37.5	72.5	16.7	13
4	26.8	57.6	16.8	8
5	27.8	58.8	19.1	21

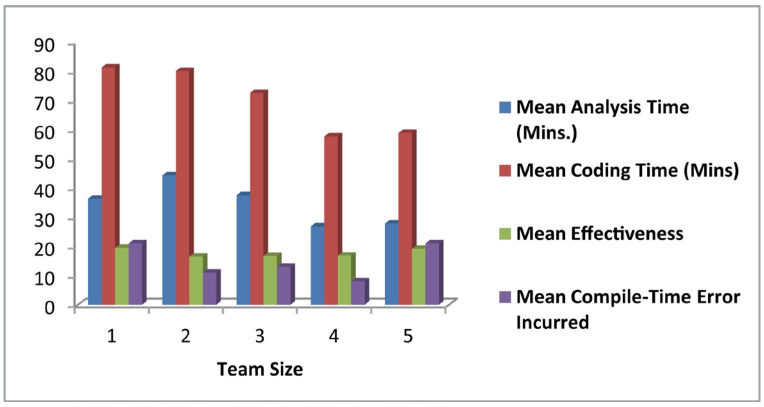


Figure 1. The performances of the teams in analysis, coding time, bugs and effectiveness.

Table 1 and **Figure 1** show that the mean analysis times taken by the teams were high for Team sizes 1, 2 and 3 with Team size 2 having the highest time. Team sizes 4 and 5 seem to have the same mean analysis time. The mean coding times were also high for Team sizes 1, 2 and 3. Team sizes 4 and 5 also had more or less same mean coding times. As for the mean effectiveness, which measured the programmers' accuracy and efficiency, Team sizes 1 and 5 had the highest value; followed by Team size 4. The margins being small for Team sizes 2, 3 and 4. Team size 4 had the least mean compile time errors incurred, while Team sizes 1 and 5 incurred the highest number of errors in the experiment.

The optimum team size was obtained with the criteria of the team that has the

1) least time spent on problem analysis;
2) least time for coding;
3) best effectiveness score;
4) least number of bugs reported.

From **Figure 1**, it can be inferred that team size 4 minimally fulfilled the above four criteria.

4.2. Experiment 2: Randomly Assigned Teams with Same Gender Compositions

The experiment was repeated using same gender team compositions of different sizes. **Table 2** and **Figure 2** give the results obtained with single male gender teams of different sizes.

For the male teams, the time for problem analysis was highest for Team size 3. Team sizes 1, 4 and 5 had low values with small marginal values. The time spent on coding was highest with Team size 2. Team size 4 had the highest effectiveness followed by Team size 3. Compile time errors incurred was very high with Team size 1 followed by Team size 5. Team size 4 had the least number of errors reported.

Using the criteria we set to determine the optimum team size, it is also inferred that the optimum team size obtained with male teams is 4.

Table 3 and **Figure 3** give the results obtained with single female gender teams of different sizes.

Table 2. Male teams' performances.

Team Size	Analysis Time (hrs)	Coding Time (hrs)	Effectiveness	Compile-Time Error (Bugs) Incurred
1	0.42	3.44	18.0	59
2	0.67	4.67	18.0	10
3	0.87	3.98	24.0	6
4	0.42	3.14	26.0	3
5	0.40	4.05	14.0	28

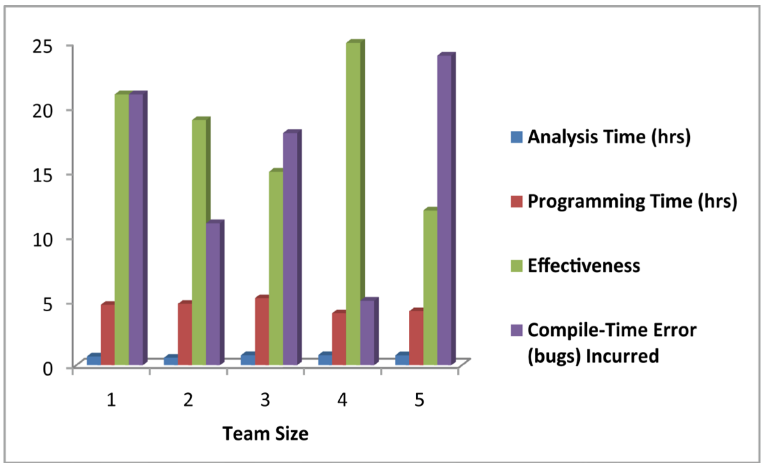


Figure 2. Male teams' performances.

Table 3. Female teams' performances.

Team Size	Analysis Time (hrs)	Coding Time (hrs)	Effectiveness	Compile-Time Error (Bugs) Incurred
1	0.68	4.68	21	21
2	0.58	4.76	19	11
3	0.78	5.20	15	18
4	0.78	4.03	25	5
5	0.77	4.19	12	24

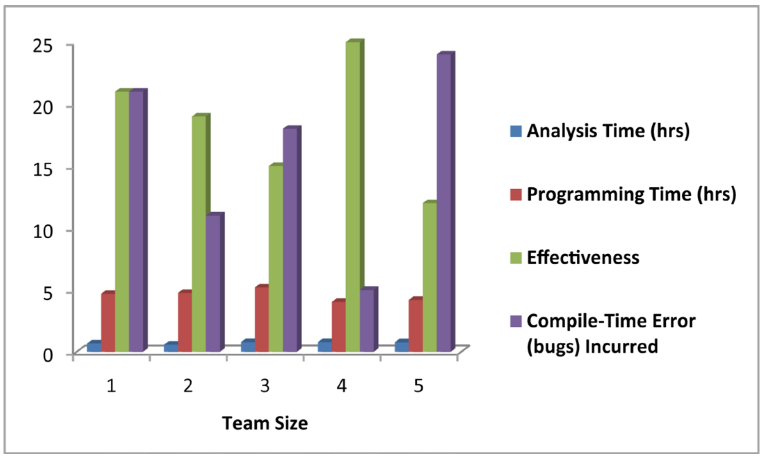


Figure 3. Female teams' performances.

With female teams, Team sizes 3, 4 and 5 had high time spent on problem analysis. Team size 4 had the least time spent on coding and highest effectiveness. The team size also had the least number of bugs reported. The result shows that Team size 4 also performed at optimum level in the experiment.

4.3. Discussion of Results

Collaborative learning is increasingly being recognized as a technique for improving learning outcomes [17]. In collaboration students actively participate in problem solving processes by communicating about the conceptual representations relating to the task at hand. Today, employers are demanding that students have teamwork skills. Past results from the US show that while most graduates have satisfactory technical skills, their teamwork abilities are often deficient [18] [19].

The task of developing quality software requires a lot of analysis, design and programming which also requires very careful design. The various tasks to be done during software development have resulted in obvious division of persons into analysts, designers and programmers. Collaborative learning involves grouping or pairing students to work together. This practice has been nominated several times in the last decades as an improved way of developing software [11].

One of the most difficult decisions in programming teamwork or collaboration task in Computer Science is how to determine how many people should be in a team. The current research in this area has not been précised on the appropriate team (group) size. Various factors such as size of people participating affect the effectiveness of the collaboration. Dillenbourg [5] study about efficiency in collaborative situations listed group composition, group size and individual differences between group members as conditions that could affect collaboration. Large teams have always been considered unwieldy and ineffective in delivering results while small teams are perceived to be a lot more sure footed and better at delivering results. Most experts advocate that the ideal team size should lie between 5 - 10 individuals [13] while Le Jeune [7] argued in favour of small-group interactions.

This study was designed for two purposes. First, to inculcate and acquaint the students involved in the study with the spirit of team work in software projects. The second reason was to empirically determine the effective (optimum) team size that may be desirable in learning and in a real life programming environments. To achieve these objectives, two different experiments were set up in this study. Parameters for determining the optimal team were listed. The parameters are that an optimal team should spend less time in problem analysis and coding, less number of errors should be reported by the team and the team's effectiveness (efficiency and accuracy) must be superb.

Results from the two experiments conducted indicate that teams of four people working collaboratively on software projects given were able to perform effectively at optimal level. This result could be attributed to the fact that collaboration with four people improves the analysis and programming skills of the participants, thereby reducing the time spent on the task. With one of them acting as the coordinator for the team, another in charge of typing on the system; others can act as inspectors for bugs and wrong coding styles as the task progresses. Two heads are better than one, so the maxim says. But it is demonstrated in this study four good programmers will make an optimal team in computer programming tasks.

Our results on optimum Team size 4 are very much in line with the proposal of Institute for the Management of Information Systems' 2013 Diploma Syllabus that says "*Try to get students to discuss case study work in teams. Get them to ask* '*what if*' *type questions about the models they are considering. A group size of four seems to be optimal*" [20]. Katzenbach and Smith [21] also show that teams are far more difficult to form. However, they fare better as *real teams* when they are small. Stephen [22] opines that teamwork principles of mutual accountability and cohesiveness that are necessary to achieve high performance become difficult in large teams.

5. Conclusion

Two independent experiments conducted in this study demonstrate that students who are new to computer programming can benefit a lot when the teaching/learning is done in a collaborative environment. The study reveals that the optimal team size that may be adopted in learning/programming environment should be four. This will enhance effectiveness of the members as well as reduction in the time spent on problem analysis and coding as well as the number of bugs that may be incurred in programming.

Acknowledgements

We specially acknowledge the 200 level students of the Department of Computer Science, University of Ibadan, Nigeria for their sincere and faithful participation in this study. The study was carried out as part of their regular teaching and practical periods in the first semester of 2012/2013 session.

References

[1] de Faria, E.S.J., Adán-Coello, J.M. and Yamanaka, K. (2006) Forming Groups for Collaborative Learning in Introductory Computer Programming Courses Based on Students' Programming Styles: An Empirical Study. 36*th ASEE/IEEE Frontiers in Education Conference*, San Diego, 28-31 October 2006, 6-11.
http://fie2012.org/sites/fie2012.org/history/fie2006/papers/1170.pdf

[2] Bryant, S., Romero, P. and du Boulay, B. (2006) The Collaborative Nature of Pair Programming, Extreme Programming and Agile Processes in Software Engineering. *Lecture Notes in Computer Science*, **4044**, 53-64.

[3] Müller, M.M. (2005) Two Controlled Experiments Concerning the Comparison of Pair Programming to Peer Review. *Journal of Systems and Software*, **78**, 166-179. http://dx.doi.org/10.1016/j.jss.2004.12.019

[4] Swamidurai1, R. and David, A. (2012) Collaborative-Adversarial Pair Programming. *International Scholarly Research Network ISRN Software Engineering*, **2012**, 1-11. http://downloads.hindawi.com/journals/isrn/2012/516184.pdf

[5] Dillenbourg, P., BAKER, M., Blaye, A. and O'Malley, C. (1996) The Evolution of Research on Collaborative Learning. In: Spada, E. and Reiman, P., Eds., *Learning in Humans and Machine*: *Towards an Interdisciplinary Learning Science*, Elsevier, Oxford, 189-211.

[6] World Health Organization (WHO) (2007) Team Building. WHO, Geneva.
http://www.who.int/cancer/modules/Team%20building.pdf

[7] LeJeune, N. (2003) Critical Components for Successful Collaborative Learning in CS1. *Journal of Computing Sciences in Colleges*, **19**, 275-285.

[8] Zimmermann, T. and Bird, C. (2012) Collaborative Software Development in Ten Years: Diversity, Tools, and Remix

Culture.
http://www.researchgate.net/publication/236644417_Collaborative_Software_Development_in_Ten_Years_Diversity_Tools_and_Remix_Culture

[9] Chaparro, E.A., Yuksel, A., Romero, P. and Bryant, S. (2005) Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education. 17*th Workshop of the Psychology of Programming Interest Group*, Sussex University, June 2005, 5-18. http://www.ppig.org/papers/17th-chaparro.pdf

[10] Begel, A. (2008) Pair Programming: What's in It for Me? http://dl.acm.org/citation.cfm?id=1414026.24th

[11] Williams, L. (2008) Pair Programming. North Carolina State University.
http://collaboration.csc.ncsu.edu/laurie/Papers/ESE%20WilliamsPairProgramming_V2.pdf

[12] Mendes, E., Al-Fakhri, L.B. and Luxton-Reilly, A. (2005) Investigating Pair-Programming in a 2nd-Year Software Development and Design Computer Science Course. *Proceedings of ITiCSE*'05, Lisbon, 27-29 June 2005, 1-5.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.4157&rep=rep1&type=pdf

[13] Team Building (2009) Team Performance and Team Size. www.teambuildingportal.com

[14] The Free Encyclopedia (2011) Programming Team. http://en.wikipedia.org/wiki/Programming_team

[15] Bryman, A. and Cramer, D. (1997) Quantitative Data Analysis with SPSS for Windows. Loughborough University, Loughborough.

[16] Porter, A.A., Votta Jr., L.G. and Basili, V.R. (1995) Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE Transactions on Software Engineering*, **21**, 563-575.
http://dx.doi.org/10.1109/32.391380

[17] Beck, L.L., Chizhik, A.W. and McElroy, A.C. (2005) Cooperative Learning Techniques in CS1: Design and Experimental Evaluation. *Proceedings of the* 36*th SIGCSE Technical Symposium on Computer Science Education*, St. Louis, 23-27 February 2005, 470-474. http://dx.doi.org/10.1145/1047344.1047495

[18] Waite, W.M., Jackson, M.H. and Diwan, A. (2004) The Conversational Classroom. *Proceedings of the* 34*th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, 3-7 March 2004, 127-131.

[19] Bower, M. and Richards, D. (2006) Collaborative Learning: Some Possibilities and Limitations for Students and Teachers. Computing Department Macquarie University.
http://www.ascilite.org.au/conferences/sydney06/proceeding/pdf_papers/p150.pdf

[20] The Institute for the Management of Information Systems (2013) Diploma Syllabus. Version 2, 46.
http://www.bcs.org/upload/pdf/imis-diploma-syllabus.pdf

[21] Katzenbach, J.R. and Smith, D.K. (2005) The Wisdom of Teams: Creating the High Performance Organization.
http://www.amazon.com/The-Wisdom-Teams-High-Performance-Organization/dp/0060522003

[22] Robbins, S.P. (2005) Essentials of Organisational Behaviour. 8th Edition, Prentice Hall, Upper Saddle River.

Appendix

Experiment 1

Aim: To determine the optimum team size among various parallel teams working collaboratively in programming.

The Problem: Implement a Java code for CSC Grammar School that has just concluded an entrance examination test for her prospective students. This has been a yearly routine exercise for the intending students of the school. As a matter of fact, varied number of students attends the test yearly. The range of scores in the test always varies from 0 to 80; 50 being set as the cut-off point for the admission. The management of the school has just decided to employ the use of computers to assist them in computing some statistics from the yearly test results and you have been contacted on this, being a prospective full-fledged computer programmer.

Following information will be needed from the examination results data:

1) Total number of students that participated in an entrance examination test;

2) Number of students that are admit-able in a session;

3) The average score in the examination;

4) The maximum and minimum score in the examination;

5) Total number of students that fail the test;

6) Total number of students in the marginal score levels (between 48 and 49 inclusive) for possible consideration in the second batch;

7) Total number of students in the following score categories:

A. 0 - 20

B. 21 - 40

C. 41 - 49

D. 50 - 80

Task

1) Implement a java program, using the concepts of OOP to read the data of about 20 students or more and computes all the information needed by the school;

2) Your outputs, should be generated with appropriate sectional information headings.

Experiment 2

Aim: To determine the optimum team size in programming when participants of same gender work together in teams.

Task: Data sent across Computer Networks has always been vulnerable to so many types of security threats. Anyway, data encryption algorithms have been developed over the years, with some relative problems associated with them. Akinola (2009) develops a data encryption algorithm recently, with the following features:

1) A data to be sent across a network is taken as a single string. However, the minimum number of characters in a string has to be three (3);

2) The string is then divided into two roughly equal substrings, A and B;

3) Each of the substrings A and B are then encoded in a reverse order;

4) To further encrypt the string data, vowels in the substrings are encoded as follows:

Vowel	Encoding Number
a	1
e	5
i	3
o	2
u	4

5) To send the original string data from its source, the encrypted substrings A and B are concatenated with A coming after B;

6) At the receiving end, Steps (2) and (3) are repeated on the sent encrypted string, in order to decrypt it. All the encoding numbers in Step (4) are converted to their respective characters in the received string.

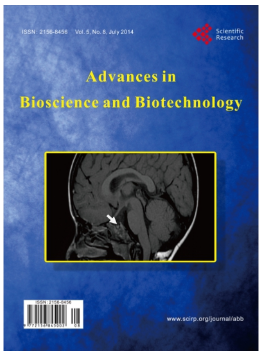
Tasks

1) Implement a **Java class** named **AkinCrypt** for the Akinola's encrypting algorithm;

2) Methods to be included in your class should be the following; however, you are free to add more relevant methods:

- Method to return the encrypted string to be sent on the network;
- Method to return the decrypted string at the final destination.

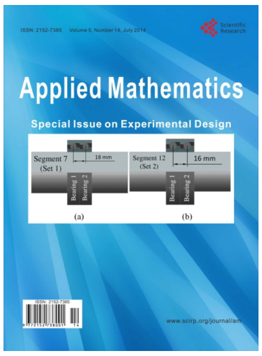
3) Implement a tester program for the class.


Advances in Bioscience and Biotechnology

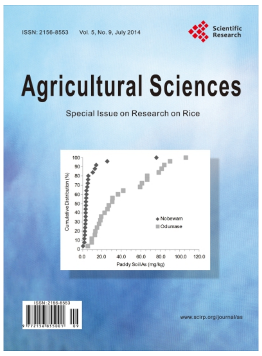
American Journal of Analytical Chemistry

American Journal of Plant Sciences

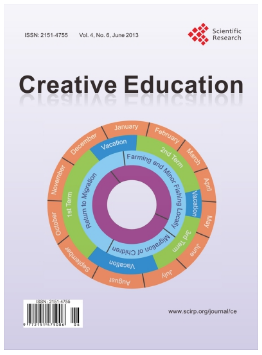
Special Issue on Photosynthesis

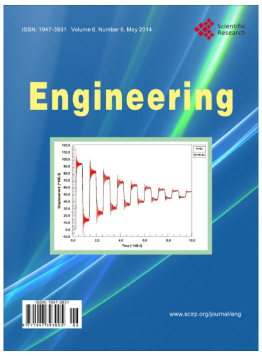

Applied Mathematics

Special Issue on Experimental Design

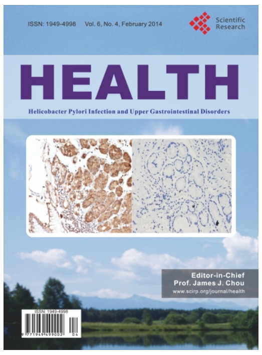

Agricultural Sciences

Special Issue on Research on Rice


Creative Education


Engineering

Food and Nutrition Sciences


HEALTH

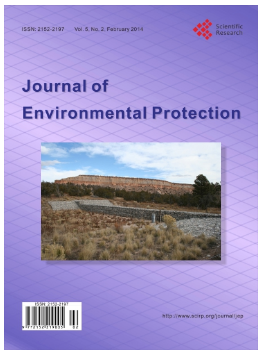
Helicobacter Pylori Infection and Upper Gastrointestinal Disorders

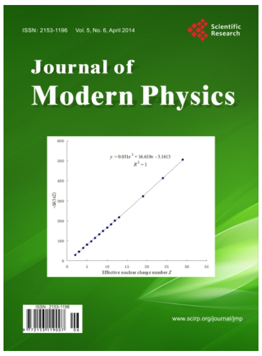
Editor-in-Chief Prof. James J. Chou

Journal of Computer and Communications

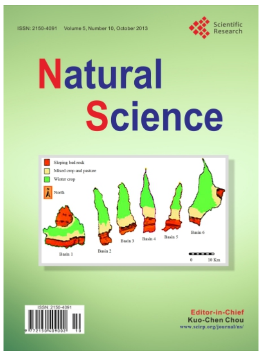
Journal of Cancer Therapy

Special Issue on Bone Tumor Research

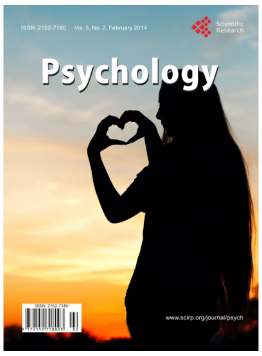

Journal of Environmental Protection


Journal of Modern Physics

Modern Economy


Natural Science

Editor-in-Chief Kuo-Chen Chou


Psychology