Data Analysis for finding the Efficiency in different Programming Language Courses

Dr Anuradha.S.G
Associate Professor
Department of Computer and Engineering
RYMEC, Ballari -583101, Karnataka (INDIA)
anuradhasuresh13@rymec.in

Abstract

Ensuring the outcomes of Programming language courses is a taxing process in the academic environment especially during the early stages of collegiate education. Attainment of outcomes is highly influenced by levels of learners and pedagogy. Students' level of learning differs due to several factors such as the background, medium of instruction, logical thinking, creativity and associating problems with solutions. Students need to develop programming skills for employability but the current pedagogy ignores levels of learners and their abilities in the teaching learning environment. In this regard, the authors conducted two experiments to discover the gaps based on the problems faced while teaching/learning of programming and accomplishment of learning outcomes. The results confirm that students lack higher order thinking skills and hence there is a need to update the teaching learning environment by using suitable ICT-enabled systems to teach students the programming language courses so as to ensure they achieve the learning outcomes.

Keywords*: Outcome based education, Pedagogy, Higher order thinking, Bloom's Taxonomy, learning outcomes, ICT*

1. Introduction

Problem solving using Computer programming languages is fundamental to any study relating to computer science. Having been in the field of teaching for more than fifteen years, a general observation is that there is a decrease in the number of students who take up profession as software engineers at the end of three years. There is also a considerable decrease in women students wishing to master programming or taking up a job in the IT industry. A case study on this traced the reason to students lacking analytical skills and the inability of the faculty members to cater to a large class of 25 students and above (in India) as against 1:15 as noted by the International Higher Education system. As a result, in majority of the institutions, the ethnicity of inquiry and analysis cannot be indoctrinated as component of higher education informs the Education Quality Upgradation and Inclusion Programme (EQUIP), that targets principles of access, inclusion, quality, excellence, and enhancing employability in higher education.

Students are exposed to a programming language right from the beginning of their undergraduate program. Initially every student struggle to do programming but gradually as the days pass, they learn the syntax and semantics of the programming language and try to apply it in problem solving. But this is not so with all students. In majority of the cases, within a few months of beginning a course in a programming language, the student concludes that programming is difficult and loses interest in the course. Winslow (1996) observes, novice programmers go through the program line by line. However, the expert programmers acquire in depth knowledge on programming and work with blocks of code. Perkins et al. (1986) categorizes students into three groups namely stoppers, movers and extreme movers. Stoppers are students who lack focus and when challenged with a problem, they just give up, while movers are a group of students who use feedback about errors to move on with programming making better progress in their work. The third category is extreme movers also called tinkerers, who make changes randomly and like the stopper group, show no progress in their work. Applying this categorization in the context of students of the authors' institution in Tirupattur district of Tamil Nadu,

India, the authors found the ratio as 6: 3:1. Majority of the students fall in Stopper Category And do not know how to solve problems. One of the possible reasons is due to deficiency of mathematical and analytical skills (Koulouri et al., 2015) or because the staff would have focused on teaching their programming concepts rather than developing problem solving abilities.
The organization of this paper is as follows. Section 2 contains related literature. Section 3 describes the experiment done with methodology applied, the description of the data set, question design and assessment. There are two experiments conducted. The first is a survey on focal areas that make it difficult to learn programming languages while the second is a programming aptitude test based on Bloom's Taxonomy. Section 4 presents the results and discussion and Section 5 is the conclusion.

2. Early Work

Many educational researches have been carried out to identify the distinctiveness of novice programmers and to study the challenges in teaching programming and solution approaches to teach the same. Robins et al. (2003) observes that novice programmers have limited and superficial knowledge, lack hypothesis and fail to apply the acquired knowledge on new problems, while the experts do have sophisticated knowledge with problem solving abilities. Brooks (1983) shares a similar opinion where he notes that experts formed hypothesis while the novices did not. Many times, the novice programmers hold misconceptions and misunderstandings. Robins suggests that novices should be given course materials that are simple to understand and follow, which at a later stage can be extended slowly as the novices gain experiences. DuBuolay (1986) points out that the first problem with novices is that they fail to understand what a program is useful for and the benefits of learning programming. The other difficulties include the representation of the language with its syntax and semantics, understanding the structure and inability to specify, develop, test and debug a problem using tools available.

Hromovic observes that programming is a skill to communicate. When students are taught programming, they must move from simple to complex structures. Some researches focus on the programming language to be introduced. Since the students get anxious at learning programming in the beginning, few authors suggest that a simple programming language could be introduced as an introductory language. In a study on the practices, difficulties and opportunities, the authors found that the majority of both the surveyed students and staff preferred to have object-oriented programming as an introductory programming course. Koulouri et al. (2015) suggest that Python be introduced as the introductory language as it is simpler when compared to the procedural languages. Initially novice programmers make a lot of mistakes in syntax and structure. Neither will they be able to complete the code in the given time. To overcome this, Ali &Smithin (2014) introduced Alice programming language as the environment helps students to code without having to worry about semicolons, curly braces etc. and allow the students to focus on the concepts.
Research has also been carried out to identify the difficulties of learners relating to course content. Garner et al. (2005) in their study assert that most novices face problems with Loops, Constructors and control structures. Lahtinen states that novices have difficulty in understanding pointers, advanced data structures, references and using the language libraries. Milne and Rowe (2002) add polymorphism, operator overloading, dynamic allocation of memory and pointers to the list. Goldman et al. includes problems with Inheritance which is also supported by Schulte and Bennedsen(2006).
Another common problem is the lack of mathematical and analytical skills in novices as showcased by the majority of the researchers including Sandra Milena, Gomes and Anabella, Robins. Due to the lack of problem-solving skills, students may be involved in plagiarism and write the code without understanding it. Researchers Ali and Smithin suggest that before students learn the programming language, they should go through a preliminary course that will prepare them for the programming language education. To encourage analytical thinking, Dijkstra (1968) proposes a curriculum based on mathematical foundation such as Predicate Calculus, Boolen Algebra and establishing formal proofs of programming correctness.

Gomes and Anabella observe that staff members are focused on teaching syntax rather than problem solving skills. So the author suggests that the students are taught problem solving rather than syntax. To personalize teaching, the authors also suggest that a computing teacher could be used where the teacher would not show negative sentiments rather exhibit patience and would have different presentation format for each activity.
There are also non-course related difficulties such as students' previous learning experience and students' psychological facet. Oroma et al. (2012) observes that apart from a simple introductory programming language student should also be intrinsically motivated to learn programming. As Bandura (1994) points out students with self-efficacy always undertake taxing tasks, take special efforts to accomplish them and exhibit perseverance when difficulties arise, but not all students have this quality of self-efficacy.
Programming is a multiprocessing task as noted by Jenkins (2002). During the process of programming, problems are identified, analyzed and a specification must be translated into an algorithm.

3. Experiment and Methodology

Based on the early work done in the field, two experiments were conducted to find out the possible reasons and propose a solution approach.

3.1. Focal areas Survey

Considering the areas of importance as conceived by Pears et al. (2007), the first experiment was a survey on the focal areas of curriculum, pedagogy and tools used by students and staff of selected colleges. Hoping to get the accurate reasons for difficulty and for taking corrective action, questions were distributed based on the importance of each of the factors.

- Questions design and Assessment

The students had 20% of the questions based on the curricular aspects, 40% based on pedagogy used for teaching, 16% of the questions were based on choice of language selected for teaching, 12% based on tools used for teaching and 12% on the literature available / used for learning programming. Similarly, the questionnaire used for faculty had 18% of questions based on curricular aspects, 36% based on pedagogy, 14% on choice of language, 7% on tools used for teaching, 14% on the psychology / Attitude of students and the rest 11% on the literature available.
These questions had five options namely 5 – YES / 4 – PARTIALLY YES / 3 – PARTIALLY NO / 2 - NO / 1 – UNABLE TO DECIDE.

3.2. Programming Aptitude Test

The second experiment was a Programming Aptitude Test for the course ***Computer Graphics & Visualization using OpenGL*** that aimed at predicting the knowledge levels of the students based on Bloom's Taxonomy. The objective was to identify areas where students lacked cognition and to help them develop higher-order thinking.

- Questions design and selection of dataset

Initially, two sets of questions were used for the test. Set one had questions from OpenGL Programming and this was used for the third-year students, while set two was also designed on ***Computer Graphics & Visualization*** for the third-year students of department of Computer Science and Engineering Rao Bahadur Y Mahabaleshwarappa Engineering College - [**RYMEC**], Bellary, Karnataka. This analytical test was further extended to other colleges, with a change in the question paper. The questions were based on C++ programming language and were framed in a way to test the skills of students in every phase of the taxonomy. The test was conducted on the third-year students of four rural and semi-urban colleges and two urban colleges. The study group was chosen to discover the distinction (if any) in the comprehension or application of knowledge by students of rural or urban areas.

4. Results and Discussion

An analysis of the responses of the first survey on Focal areas, shown in table1, depicts that most problems such as tools used for teaching, pedagogy, curricula, the attitude etc arose from the fact that most of the students who chose computer science as their program of study were first generation learners hailing from rural areas. These students had gone through their higher secondary education with the medium of instruction being in the local dialect and once in college, they found it difficult to cope with English because of which they were memorizing rather than understanding the program code. Since the majority of them were first generation learners, they also lacked the family accompaniment in their studies. The pictorial representation of responses is shown in figure1.

Table1: Responses of staff and students in percentage

FACTOR	Staff	Student	G.Mean
Curricula	81.5	79.6	80.54
Pedagogy	43	69	54.47
Language Choice	83	84.2	83.59
Tools for teaching	85.5	89.4	87.42
Literature	15	11.4	13.07
Attitude	9	50	21.21

The curriculum of the Undergraduate program in colleges is mostly decided by the university. The authors observed that an average of 81% of the surveyed students and staff felt the need for a change in curriculum. 56% of students and staff felt the need for an improvement in the pedagogy. They felt more tutorials, workshops, audio visuals could be used along with an increased number of class tests topic wise. Almost 83% of both students and staff acknowledged that use of OpenGL as introductory programming interface while the rest 17% felt that other languages could be used. An average of 88% of both the students and staff felt that more audio visuals, online tutoring systems and E-learning platforms could be used. Only an average of 14 % of the surveyed people felt that there were enough resources made available while the majority felt there could still be more resources (Customized) offered to the students. Only 9% of the staff felt the students were motivated and enthusiastic about learning programming languages, while 50% of the students claimed that they were self-driven and interested in programming. The responses of students and staff on focal areas are shown in Figure1. The X-axis denotes the focal areas while the Y-axis denotes the values assigned.

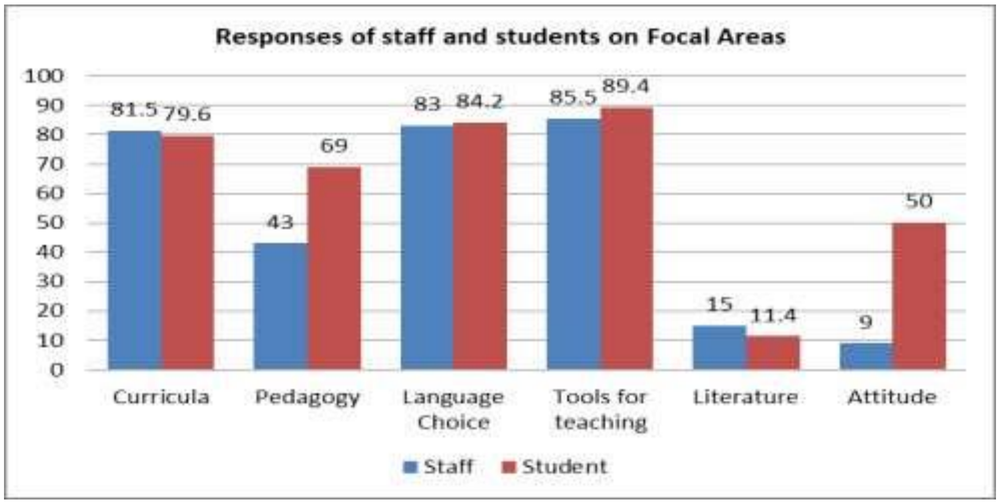

Figure 1: Analysis of Responses by Staff and Students

Analyzing the second experiment of Programming Aptitude Test based on Bloom's Taxonomy, the authors found that the students scored higher in the lower order thinking skills such as remembering, and understanding but scored lesser in higher order thinking such as analysis, evaluation and creation. When a programming exercise is given, only a few understand and write the code. There are a majority of students who plagiarized programs due to lack of analytical and critical thinking skills to work out a solution. As Robins et al. (2003) observes majority of the novice programmers had incomplete and shallow knowledge, lacked conceptual models and were unsuccessful in applying knowledge The skills required to think critically include examination, elucidation, manifestation, evaluation, deduction, rationalization, problem solving, and decision making but the students lacked all of these cognitive skills. They are so used to memorizing the answers including mathematics that they are unable to identify or comprehend the problem. Students are unable to break down a problem into smaller steps to solve it.
The Geometric Mean of the scores acquired by students of both rural and urban areas is shown in Table 2 and its pictorial representation is given in Figure 2. On the X-axis of the figure is given the various phases of Bloom's Taxonomy while the Y-axis shows the values assigned.

Table 2 - Performance of students in Programming test based on Bloom's Taxonomy [Values shown in percentage]

Colleges in Areas	Phases of Bloom's Taxonomy					
	Remember	Understand	Apply	Analyze	Evaluate	Create
Rural	66.90	25.37	10.31	9.70	16.15	3.87
Urban	88.43	36.80	32.66	14.08	39.40	1.71
G.Mean	76.91	30.55	18.35	11.68	25.22	2.57

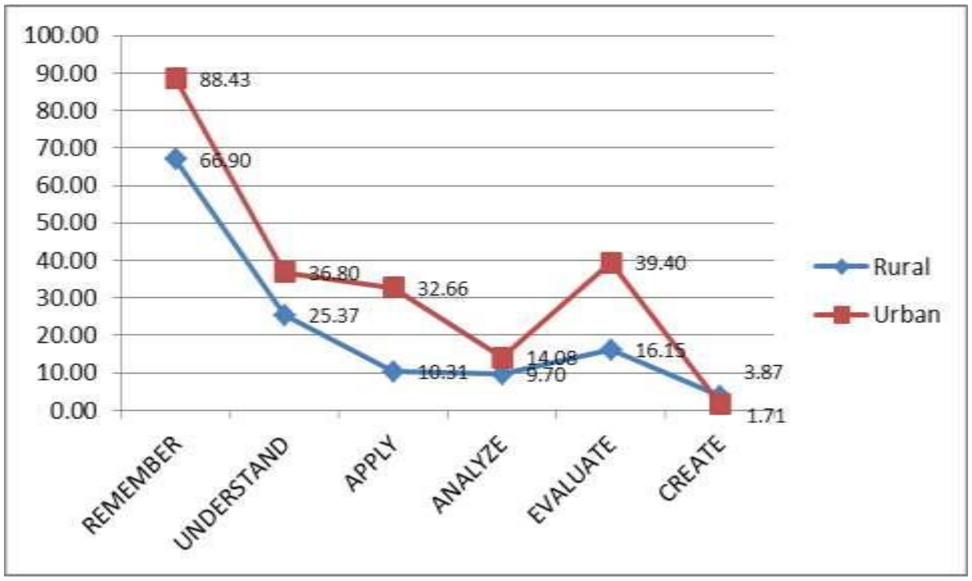

Figure 2 - Performance of students in Programming test based on Bloom’s Taxonomy

Since the focal areas directly or indirectly affect every phase of learning, it becomes necessary to relate the two. Fuzzy Topsis technique was found to be a promising method in combining the two. Hwang and Yoon (1981) was the first to propose the Technique for Order Preference by Similarity to the Ideal Solution (TOPSIS). The fuzzy TOPSIS technique embeds the priori weights. The core of the ranking for this method lies in the distance of alternatives to the ideal and anti-ideal solutions. Many authors have used the fuzzy TOPSIS method as a decision-making method by Chen (2000), Chu and Lin (2003), Jahanshahloo et al. (2006), Wang and Chang (2007), Gumus (2009) and Kilic et al., (2014)

The following illustration depicts the decision-making approach:

Step 1. First the Geometric Mean of both the survey and programming test scores were derived. Using Pythagoras formula to combine different data sets, the Pythagoras Decision Matrix was derived as follows:

Table 3- Pythagoras Decision Matrix

Decision Matrix	Remember	Understand	Apply	Analyze	Evaluate	Create
Curricula	111.37	86.15	82.61	81.39	84.40	80.59
Pedagogy	94.25	62.45	57.48	55.71	60.03	54.53
Language Choice	113.60	89.01	85.59	84.41	87.32	83.64
Tools for Teaching	116.45	92.61	89.33	88.21	90.99	87.47
Literature	78.02	33.24	22.53	17.54	28.41	13.33
Attitude	79.79	37.20	28.05	24.22	32.96	21.37

Step 2. Using fuzzy normalization formula, we calculated the normalized Decision
Matrix Table 4 - Normalized Decision Matrix

Decision Matrix	Remember	Understand	Apply	Analyze	Evaluate	Create
Curricula	1.0000	0.7735	0.7417	0.7308	0.7579	0.7236
Pedagogy	1.0000	0.6627	0.6098	0.5911	0.6369	0.5786
Language Choice	1.0000	0.7835	0.7534	0.7431	0.7687	0.7363
Tools for teaching	1.0000	0.7953	0.7672	0.7575	0.7814	0.7511
Literature	1.0000	0.4260	0.2888	0.2248	0.3642	0.1708
Attitude	1.0000	0.4662	0.3515	0.3035	0.4131	0.2678

Step 3. From this normalized decision matrix, fuzzy weighted normalized decision matrix is derived by first assigning weightage to every phase of learning and the weighted vector values of the learning attributes are $(0.0833, 0.1250, 0.1250, 0.2083, 0.2083, 0.2500)^T$ respectively. Then multiplying the element of normalized decision matrix by the weightage assigned.

Table 5- Fuzzy weighted normalized decision matrix

Decision Matrix	Remember	Understand	Apply	Analyze	Evaluate	Create
Curricula	0.0833	0.0967	0.0927	0.1522	0.1579	0.1809
Pedagogy	0.0833	0.0828	0.0762	0.1231	0.1327	0.1446

Language Choice	0.0833	0.0979	0.0942	0.1548	0.1601	0.1841

Tools for teaching	0.0833	0.0994	0.0959	0.1578	0.1628	0.1878
Literature	0.0833	0.0532	0.0361	0.0468	0.0759	0.0427
Attitude	0.0833	0.0583	0.0439	0.0632	0.0861	0.0670

Step 4. Once the weighted normalized decision matrix is formed, ranking of factors is done. In any ranking method, the first step is to identify the ideal solution I^+ and the anti-ideal solution I^-. The Ideal solution (I^+) is the best score in all criteria while the Anti-ideal solution (I^-) is the worst score. The Ideal solution in our case is 1.000 while the Anti-Ideal solution is 0.00. The distance from each alternative solution to the positive ideal solution is calculated using the Euclidean distance as

$$dT_i^+ = \sqrt{\sum_{j=1}^{m} w_j^2 \left(v_j^+ - v_{ij}\right)^2}, \quad i = 1, \dots, q.$$

Therefore, the solutions of dT^+i= (0.3679, 0.3796, 0.3669, 0.3657, 0.4100, 0.4034) (i= 1,2, 3…,6). In turn, the distance from an alternative solution to the anti-ideal solution is calculated by formula

$$dT_i^- = \sqrt{\sum_{j=1}^{m} w_j^2 \left(v_j^- - v_{ij}\right)^2}, \quad i = 1, \dots, q.$$

Similarly, the solutions of dT^- = (0.1861, 0.1889, 0.1859, 0.1857, 0.1998, 0.1970) (i= 1,2, 3…,6).

$$D_i^+ = \frac{dT_i^-}{dT_i^+ + dT_i^-}$$

Finally, the closeness coefficient is calculated, to determine the ranking order of all factors, as follows:

The largest value of D i+ is accepted as the best solution, while the smallest value is regarded as the worst solution. The ranking of factors is depicted in the following table:

Table 6 - Ranking of Factors

Factors	Curricula	Pedagogy	Language Choice	Tools for teaching	Literature	Attitude
Closeness Coefficient	0.3360	0.3323	0.3363	0.3368	0.3276	0.3281
Rank	3	4	2	1	6	5

The analysis of the scores proves that students' performance will be better if programming is taught using appropriate tools accompanied by innovative teaching practices.

5. Conclusion

There are varied reasons which make it difficult for the learner to use programming languages for problem solving. Most common reasons could be lack of analytical or problem-solving skills, inappropriate teaching pedagogies and tools and lack of intrinsic motivation among students. To ensure students apply programming languages for problem solving, the issue of thinking and analytical skills needs to be specially addressed. To encourage cognitive thinking and learning, the authors are working on an ICT model that encompasses activities for different levels of learners so as to ensure the accomplishment of the learning outcomes of programming courses.

References

1. Nathan Rountre, Anthony Robins and Janet Rountree (2003), "Learning and Teaching Programming - A Review and Discussion", Computer Science Education, Volume 13, No: 2, pp 137 to 172,http://dx.doi.org/10.1076/csed.13.2.137.14200
2. Albert Bandura (1994), "Self Efficacy", in V. S. Ramachaundran (Ed), Encyclopedia of Human Behaviors, Vol 4,https://doi.org/10.1002/9780470479216.corpsy0836
3. Arnold Pears, Stephen Seidman, MalmiLauri., Mannila Linda, Adams Elizabeth, Bennedsen Jens, Devlin Marie, Paterson James (2007), "A survey of literature on the teaching of introductory programming", Working Group Reports on on Innovation and Technology in Computer Science Education, doi:10.1145/1345443.1345441
4. Azad Ali and David Smith (2014), "Teaching an Introductory Programming Language in a General Education Course", Journal of Information Technology Education - Innovations in Practice, Volume 13 , pp 057 to 067,https://doi.org/10.28945/1992
5. Boulay, Benedict Du (1986), "Some Difficulties of Learning to Program", Journal of Educational Computing Research, Volume 2, Issue 1, pp 57 to 73, doi:10.2190/3lfx-9rrf-67t8-uvk9
6. Canran Liu, Frazier Paul, Lalit Kumar, Macgregor Catherine& Blake Nigel (2006), "Catchment- wide wetland assessment and prioritization using the multi-criteria decision-making method TOPSIS. *Environmental management"*, *38*(2), pp 316-326, https://doi.org/10.1007/s00267-005- 0151-0
7. Chentung Chen. (2000), "Extensions of the TOPSIS for group decision-making under fuzzy environment - Fuzzy sets and systems," Volume 114, Issue 1, pp 1 to9
8. Chu, T. C. and Lin, Y. C. (2003), "A fuzzy TOPSIS method for robot selection", the International Journal of Advanced Manufacturing Technology, Volume 21, Issue 4, pp 284 to 290, doi.org/10.1007/s001700300033
9. Dijkstra, E. W. (1968), "A constructive approach to the problem of program correctness", BIT, Volume 8, Issue 3, pp 174 to186,doi:10.1007/bf01933419
10. FlorentinSmarandache(2004), "A Geometric Interpretation of the Neutrosophic Set - A Generalization of the Intuitionistic Fuzzy Set", IEEE International Conference on Granular Computing, DOI: 10.1109/GRC.2011.6122665
11. Garner Sandy, Patricia Haden and Anthony Robins (2005)," My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems", 7th Australasian conference on Computing education, Volume 42, pp 173 to180
12. Mendes A. J, Gomes, Anabela,"Learning to program - difficulties and solutions", 2007, Coimbra, Portugal, International Conference on Engineering Education(ICEE), Coimbra,Portugal
13. Gumus, A. T. (2009), "Evaluation of hazardous waste transportation _rms by using a two-step fuzzy-AHP and TOPSIS methodology - Expert Systems with Applications," 36:4067 -4074
14. HromkoviˇC, J. (2006), "Contributing to general education by teaching informatics", 2006, ISBN:3-540-48218-0978-3-540-48218-5
15. Hwang C. L. and Yoon, K. (1981), "Multiple Attribute Decision Making: Methods and Applications - A State of the Art Survey", Sprinnger-Verlag, NewYork
16. Iain Milne And Glenn Rowe (2002), "Difficulties in Learning and Teaching Programming—Views of Students and Tutors, Education and Information Technologies", 7:1,55–66
17. Jahanshahloo, G. R., Lot_, F. H., and Izadikhah, M. (2006), "An algorithmic method to extend TOPSIS for decision-making problems with interval data - Applied mathematics and computation", 175(2):1375to1384

18. Jenkins T, “On the difficulty of learning to program” (2002), 3rd Annual Conference, Loughborough University LTSN Centre for Information and ComputerScience

19. Josephat O. Oroma, Herbert Wanga, Fredrick Ngumbuke (2012),“Challenges of teaching and learning computer programming in developing countries: lessons from Tumaini university”, Proceedings of INTED2012 Conference, Valencia, Spain, https://www.researchgate.net/publication/267926470
20. Kenneth J. Goldman,Paul Gross, CindaHeeren, Geoffrey L. Herman,Lisa C. Kaczmarczyk, Michael C. Loui,Craig B. Zilles (2008), “Identifying important and difficult concepts in introductory computing courses using a delphi process: selective compression of unicode arrays in Java”, SIGCSE,DOI:10.1145/1352135.1352226
21. SelcukKilicHuseyin, SelimZaim, and DelenDursun(2014), “Development of a hybrid methodology for ERP system selection: The case of Turkish Airlines. Decision Support Systems”, Volume 66, pp 82 – 92, DOI: https://doi.org/10.1016/j.dss.2014.06.011
22. LahtinenEssi, Ala-Mutka Kirsti, JärvinenHannu-Matti (2005), “A study of the difficulties of novice programmers”, Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, DOI: 10.1145/1067445 -1067453
23. Michael Weigend (2006), “Evolution and Perspectives: the Bridge between Using and Understanding Computers”, Proceedings of the 2006 international conference on Informatics in Secondary Schools, doi:10.1007/11915355_11
24. Perkins, D. N., Hancock Chris, Renee Hobbs, Martin Fay, & Simmons Rebecca (1986), “Conditions of Learning in Novice Programmers”, Journal of Educational Computing Research, 2(1), 37–55,doi:10.2190/gujt-jcbj-q6qu-q9pl
25. Ruven Brooks (1983), “Towards a theory of the comprehension of computer programs”, International Journal of Man-Machine Studies, Volume 18, Issue 6, pp 543–554, doi:10.1016/s0020-7373(83)80031-5
26. Sandy Garner, Patricia Haden, Anthony Robins (2005), “My Program is Correct But it Doesn’t Run: A Preliminary Investigation of Novice Programmers’ Problems”, Australasian Computing Education Conference, Newcastle,Australia
27. Schulte, C., &Bennedsen, J. (2006), “What do teachers teach in introductory programming?”, Proceedings of the International Workshop on Computing Education Research, doi:10.1145/1151588.1151593
28. Shih, H. S., Shyur, H. J., and Lee, E. S. (2007), “An extension of TOPSIS for group decision making. Mathematical and Computer Modelling”, 45(7):801-813.
29. Theodora koulouri, Stanislaolauria, Robert d. Macredie (2015), “Teaching Introductory Programming: a Quantitative Evaluation of Different Approaches”, ACM Transactions on Computing Education (TOCE),doi:10.1145/2662412
30. K. Manoj, T. S. Sandeep, N. Sudhakar Reddy and P. M. D. Alikhan, "Genuine ratings for mobile apps with the support of authenticated users’ reviews," 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT), Bangalore, India, 2018, pp. 217-221.
31. T. S. Sandeep, K. Manoj, N. S. Reddy and R. R. Kumar, "Big Data Ensure Homologous Patient Enduring Therapy Time Forecast Algorithm by Healing Facility Echelon Recommendation," 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT), Bangalore, India, 2018, pp. 320-325.
32. K.Manoj Kumar, T.S.Sandeep, G.Sunil Kumar, K.Anusha, "Enhanced Text Mining Methodology in Social Media Platform," International Journal of Innovative Technology and Exploring Engineering (IJITEE), Vol. 8, no. 12, pp. 4857-4861, 2019.