# PROBLEMS IN PROGRAMMING EDUCATION AND MEANS OF THEIR IMPROVEMENT

KONECKI, M.

**Abstract:** *Programming courses are very important and challenging part of future computer experts' education process. Abstract nature of these courses however makes them rather difficult for most programming novices and these courses have rather high reported failure rates. Throughout the years the search for methods that would improve programming novices' understanding of abstract programming concepts has been conducted but it gave no generally accepted solution and the fact remains that problems of programming novices are reoccurring in every new generation. The reasons of this kind of state in programming education are analyzed and discussed in this paper. The results of conducted research about the most common problems of programming novices are also presented along with the proposed steps for improving the success rate of programming courses.*

**Key words:** *programming, courses, education, novices, problems*

**Authors´ data:** Dr. sc. **Konecki**, M[ario], University of Zagreb, Faculty of Organization and Informatics, Pavlinska 2, 42000 Varazdin, Croatia, mario.konecki@foi.hr

**This Publication has to be referred as:** Konecki, M[ario] (2014). Problems in Programming Education and Means of Their Improvement, Chapter 37 in DAAAM International Scientific Book 2014, pp.459-470, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-901509-98-8, ISSN 1726-9687, Vienna, Austria
DOI:10.2507/daaam.scibook.2014.37

## 1. Introduction

Computer programs are present in almost all aspects of modern business and other everyday life aspects. Development and maintenance of these programs is of vital importance and this asks for rather large number of programming professionals with profound knowledge about programming concepts. However, it has been noted that education in this area comes with many reoccurring problems and difficulties that programming novices experience during their studies. This fact leads to relatively high failure rates which in years have created negative opinion and fear about taking programming courses.

In order to solve these problems many attempts throughout the years have been made. Nevertheless, these problems have persisted to this day. The fact remains that programming novices have problems in learning even programming languages that are designated as programming languages for beginners. Programming requires certain way of thinking and understanding of different programming concepts and structures which are hard for most programming novices to comprehend and apply in their own programming tasks.

Some other aspects that influence this kind of state in programming education are also important. There is an important question about the motivation of programming novices to learn programming (Alaoutinen & Smolander, 2010) and also the question about the appropriate learning style that programming novices require in order to understand certain concepts. There is a need to analyze current situation and methodology that is predominantly used to teach programming novices programming in order to determine the best course of action that would address reoccurring problems of programming novices.

Most common problems of programming novices along with existing efforts and discussion about current methods used in teaching programming are presented in the rest of this paper. The suggestions about the right course of action that is to be undertaken in order to solve the problems of programming novices are also given and discussed.

## 2. Learning to program

One thing that is common knowledge among all, from programming novices to programming experts and teachers is that to learn how to program is difficult and challenging task and this claim is supported by many authors (Baldwin & Kuljis, 2001; Bergin & Reilly, 2005; Gomes & Mendes, 2007; Hanks et al., 2004; Jenkins, 2002; Peng, 2010; Robins et al., 2003). General opinion is that failure rates in introductory programming courses are high as well as the dropout rates after introductory programming courses (Nikula et al., 2011; Yadin, 2011) although some authors report different results depending upon the size of course group and other factors (Bennedsen & Caspersen, 2007).

Many attempts through history of programming languages have been made in order to develop a language which would be suitable with its syntax to beginners in the world of computer code. Such languages were Smalltalk, Pascal, Basic, HyperTalk, Logo and many others (Smith et al., 2000). Nevertheless, all that history has shown is that none of these languages was suitable enough for programming novices (Smith et al., 2000) although some research shows that the selection of programming notation facilitates different programming concepts (Wiedenbeck, 1999; Wiedenbeck et al., 1999). Taking into consideration that it is quite challenging to learn human spoken language fluently and that it takes years in order to do that properly, it can very well be argued that learning programming language which is not intuitive and does not address everyday situations is much more difficult.

Many authors agree that the programming language itself, its syntax combined with logic and concepts that are prerequisite to development of real programs is the main problem itself (Smith et al., 2000). Some authors go as far as to conclude that no programming language is suitable and cannot be suitable for novices (Smith et al., 2000).

## 3. Problems of programming novices

One of the biggest problems for programming novices is that there is a huge gap between the intuitive way in which they think and the way of thinking that is suitable for computers. Human mind is far more advanced than any computer. It operates in such way that it is able to process a huge number of connections and associations in order to do or understand something. Computers can't do that. They need a clear path, clear boundaries and coverage of all possible scenarios. Don Norman stated that the gap between programming novice's way of thinking and a way that is required by computer in order for it to be able to process some instruction is as wide as Grand Canyon (Norman & Draper, 1986). He also stated that in order to remove this gap either the user has to be moved closer to the system or system must be moved closer to the user (Norman & Draper, 1986).

Most efforts in education process are aimed at bringing the user closer to the system by teaching him the complex programming concepts and syntax. Since this approach has already been recognized as difficult for programming novices (Gomes & Mendes, 2007; Smith & Webb, 2000) it is only logical to try to develop methods that would allow the system to be moved closer to the user and that would enable the user to understand it in a way that is more intuitive and natural for him. All this also supports the conclusion that the main problem is not even the programming language syntax, but rather the concepts and structures and a whole new way of thinking that is required. So, solving the problem of teaching the novices to understand this new way of thinking would consequently enable them to use programming language syntax in order to implement solutions that are developed and designed using this new way of thinking.

This fact is also supported by some authors that claim that there are certain bugs (Pea, 1986) in understanding of computer programming that are characteristic to all programming novices of all ages. These bugs are reoccurring and they are more related to the way in which a computer has to be instructed in order to do something than to design of programming languages. Programming novices all have some form of intuitive understanding of programming concepts which are based on their age, previous knowledge and experience (Pea & Kurland, 1983), but this intuitive way of reasoning seems to be the main cause of most errors. Human way of thinking is simply different from the one that computer needs in order to understand and perform some tasks.

The main ability that computers lack is the ability of analogy, association and adaption. While humans possess these characteristics, computers don't and they have to be instructed mechanically with flawless precision and rules that cover all cases that computer is expected to deal with. Basically, it can be simply said that humans are intelligent and computers are not and this is the main difference that causes collision between intuitive way of reasoning that programming novices are using and the way of thinking required in order to write proper computer programs. There are three classes of common bugs in understanding of programming concepts among novices that have been identified (Pea, 1986):

- Parallelism bug.
- Intentionality bug.
- Egocentrism bug.

Parallelism bug denotes the misguided understanding that computer can be aware of several programming lines at the same time. For example that computer can backtrack and execute some particular condition after its terms have been met regardless of its inactivity as a programming line that has been passed and finished. Intentionality bug means that programming novices often presume what a program will do based upon only a part of its code. They frequently see something that triggers some conclusion about what the program will do and they think of this conclusion as a fact so they don't interpret the rest of the code objectively but rather in the light of their formed conclusion. Egocentrism bug means that programming novices often don't give computer enough programming instructions because they presume that computer will somehow figure out what they want regardless of the code that they have written. In this state of mind novices frequently omit various important conditions or loops.

## 4. Motivation and methodology

Another important question regarding learning programming is the question of proper motivation of programming novices (Alaoutinen & Smolander, 2010) and proper methodology since it can be seen that less and less students are interested in studying computer science (Bennedsen & Caspersen, 2007) and there is also a fact that many students do not possess sufficient and expected level of programming knowledge after passing programming courses (Ford & Venema, 2010; Lister et al., 2004; McCracken et al., 2001). Most of teachers are still highly traditional and they don't use new technologies or new methods of teaching (Hu, 2004).

Research however shows that students would rather have somewhat different way of learning programming, such as learning by example (Tan et al., 2009). Programming is a skill (Jenkins, 2002) and every skill requires many hours of hard work and practice. It is imperative that students do their assignments by themselves in order to achieve sufficient level of programming skill. Students obviously know this intuitively taking into consideration their attitude in which they denote practice as a preferred way of learning (Tan et al., 2009) but knowing the right way is not always enough. There are other aspects and methods that need further research in order to develop a methodology that would address all issues that programming novices encounter.

Some authors have conducted research that investigates possible set of factors and predictors of students' success in programming courses (Fincher et al., 2006). The same predictors could be also used in a way that would help to determine the best learning approach for every student since every student has some preferred way of learning (Jenkins, 2002), although it is rather hard to make programming courses as individual as it would be needed in this kind of approach for obvious reasons which include lack of time and lecturers in today's educational systems.

Taking this into consideration it can be concluded that some form of constructivism should be used when designing programming courses. Constructivism takes the learner on an active path where he is deeply involved in the learning process and he also builds new knowledge on top of his existing knowledge (Ben-Ari, 1998). Obviously new methods that would promote this kind of learning are needed in programming courses. A research conducted among students has shown that time consumption and motivation are the most important factors in successful finishing of programming courses (Kinnunen & Malmi, 2006) so it can be concluded that aside one's abilities motivation is the most important factor that needs to be properly addressed.

## 5. Main problems in programming courses and possible courses of action

In order to determine the main problems that programming novices report and to conclude about accepts of programming courses that are most difficult to comprehend, as well as to conclude about the reasons why they occur an appropriate research has been conducted. The research has been conducted on 190 information science students at the end of their programming course lectures in order to be able to test all aspects of interest that are part of most programming courses curricula.

The students were given the questionnaire in which they had to denote which aspects of programming they recognize as most difficult for them and questionnaire in which they had to denote their experience regarding understanding of programming problems and tasks as well as regarding designing algorithms and remembering programming language syntax.

At the moment of taking the questionnaire students have already solved 10 programming tasks with all aspects that were included in the questionnaire. The results of conducted research are given in Tab. 1 and Tab. 2.

| Programming topic | Number of respondents |
|---|---|
| Linked lists | 78 |
| Sorting | 66 |
| Working with files | 64 |

| | |
|---|---|
| Data structures | 63 |
| Arrays | 61 |
| Searching Algorithms | 55 |
| Pointers | 54 |
| Namespace | 52 |
| Do..While loop | 48 |
| Functions | 47 |
| For loop | 38 |
| While loop | 34 |
| If…else statement | 31 |
| Switch statement | 27 |
| Constants | 19 |
| Variables | 11 |
| Basic concepts of object-oriented programming | 10 |
| If statement | 8 |

Tab. 1. Problems reported by programming novices

| Questionnaire item | Mean | Std. dev. |
|---|---|---|
| I have no difficulties in understanding of programming problems that are presented to me | 1.430 | 0.384 |
| When solving programming task I have difficulties in understanding the task itself | 4.208 | 0.527 |
| I have difficulties in drawing diagram or writing pseudocode of given programming task's solution | 4.412 | 0.531 |
| I have more problems in visualizing and designing conceptual solution in pseudocode than in understanding and remembering programming language syntax | 3.951 | 0.481 |
| Designing of algorithmic solutions is difficult and not intuitive to me | 4.347 | 0.392 |
| The main problem I experience is remembering programming language syntax | 2.155 | 0.349 |
| The main problems I experience refer to understanding and visualizing programming tasks and designing their algorithmic solutions | 4.034 | 0.491 |

Tab. 2. Reported experience with designing algorithms and remembering programming syntax

The results presented in Tab. 1 are consistent with the analysis of students' developed computer programs throughout the duration of the course. Pointers seem to be the point in the curriculum where the students start to get more serious problems. The first half of the curriculum seems to be less difficult because of less abstract concepts, but as things move along and a more abstract and complex concepts are introduced, students start to have more serious problems while trying to understand given examples. The results presented in Tab. 2 show that although students don't find programming languages syntax to be easy, they have more problems in understanding of given problems and in designing conceptual solutions and algorithms in some form of pseudocode.

When considering these results and existing research it can be concluded that in order to make programming more suitable for programming novices the right course of action would be to alter existing methodology and curriculum structure in order to make programming more suitable for average student's learning style and desired pace. Another point of direction would also be to increase the motivation of students by elaborating the importance of programming for their professional career. Algorithmic way of thinking and understanding of pseudocode solutions design is reported as very challenging and vital problem for students and this fact asks for a change in learning and teaching strategy which is also reported as one of the main factors of success in programming courses (Hawi, 2010). It can be concluded that in order to try to solve the problem of programming novices several steps could be incorporated into programming education. The proposed steps are:

- Introduce additional programming course prior to introductory programming course that would promote algorithmic way of thinking.
- Increase motivation of students for learning programming.
- Explain to students that programming is a skill, not merely knowledge.
- Introduce elements of constructivism into teaching process.
- Introduce learning by example.
- Introduce animation and other visualization techniques combined with interaction.
- Introduce interactive visual simulations.
- Include support for multiple learning styles.

Because of an established gap between intuitive way of thinking and an algorithmic thinking required to develop proper computer programs it would be beneficial to introduce another programming course that would deal with this difference and enable students to train themselves in this new way of reasoning. This course could simply be called "Algorithmic thinking" or "Algorithmic construction" and it would stay away from complex syntax or programming concepts that students have most problems with. Instead, it would train students to understand the process of decomposition of various problems and how to translate those problems' parts into composition of various algorithms' parts that would do what algorithms are supposed to do, cover all angles and cases and instruct the computer to do all necessary steps to solve a problem.

To promote this way of thinking means to bridge the gap most students have regarding constructing algorithms and writing various programming code. Also, this kind of course would train students practically only in a subset of most today's programming courses curricula. Only variables, selections, loops and arrays would be absolutely necessary with addition of maybe a few other concepts. In this way the students would be able to train themselves in decomposition of problems and construction of algorithms focused on understanding of their parts and meaning while being free of trying to grasp complex programming concepts and syntax that require a lot of their time which in the end results with them not comprehending the core skills of algorithmic thinking.

By introduction of more activities and gradual advancement in collecting points as well as by various formal and informal confirmations, the motivation of students for learning programming could be increased. The same effect on students' motivation could be achieved by introduction of all other proposed steps.

It is important to explain to students that programming is a skill, not only knowledge and like every other skill, it requires many hours of practice. It is therefore of great importance for students to do all their exercises themselves and to do as many exercises they are able to do in order to develop their programming skill. It is also important to notice that any particular skill including programming is developed during time so the students need to practice over a longer period of time rather than doing a large amount of exercises rapidly.

By introducing elements of constructivism and by turning the teacher into facilitator that helps students to figure programming concepts by themselves the students will have a chance to gain more profound and durable knowledge. This kind of knowledge will also be promoted by using various analogies when describing programming concepts that will help students to build new knowledge upon already existing and familiar concepts.

Since programming is a skill, learning by example is logical way of teaching that enables students to develop their skills during the lectures and connect those skills and examples with theoretical concepts while they learn on their own.

Animations and other visualization techniques are indicated as beneficial for students (Sorva et al., 2013) and they can help students to better understand abstract programming concepts and structures through increase of students' motivation to learn since students are not frustrated or scared because they cannot imagine or understand certain aspect of programming. Research also shows that best results are achieved by not merely passive animation of programming concepts but with inclusion of students in visualization process through some form of interaction (Pears et al., 2007).

Another step that can be made is to include visual simulators of existing objects that students would be able to program just as real objects, in order to make learning more interesting and to increase students' motivation to learn (Dolinay et al., 2010, 2011). Visual simulations include students into learning process, making them active and more focused. In this way students are able to understand programming concepts in a more profound way since they have a means to investigate and simulate behavior of various programming elements. In this way the motivation of students is also increased since they are not only passive listeners but active participants in the learning process.

Every student has a different and specific preferred way of learning. By introducing a greater variety of presentation styles different learning styles would be supported which would make learning easier for all students and which would decrease fear of programming along with increase of motivation because of more natural way of learning for every particular student.

## 6. Conclusion

Programming professionals are vital part of modern business world and education of programming novices is of great importance. Programming courses are an integral part of all computer and information science studies. However, rather high failure rates and persistent problems of programming novices to comprehend programming concepts and structures are reported which leads to conclusion that to learn how to program is a challenging task that asks for a lot of effort from students and from teachers. Abstract nature of programming concepts is something that students are not used to deal with and it results in students not being able to have a clear picture of these concepts. Student often tend to stop following lectures because they lose track in some point which as a result decreases their motivation and increases their fear of dealing with programming.

Many attempts to develop a programming language that would be suitable for programming novices have been made throughout the years but none of them gave any generally acceptable results which leads to conclusion that a programming languages themselves and an algorithmic way of thinking that is required in order to write programming code are the problem itself. There is a huge difference between the intuitive way in which programming novices are reasoning and the way of thinking that is required by computers in order to understand computer code and perform task accurately.

The question of motivation and appropriate learning style is another aspect that needs to be considered and programming courses should incorporate different elements from different learning styles in order to be suitable for all programming novices. This would also increase their motivation to learn programming. Various visualization techniques, interactive simulations, learning by example and other methods would also be beneficial for programming novices and they would enable them to understand complex programming concepts in an easier way. This approach would also reduce frustration and increase the motivation of programming novices. Making clear that programming is a skill and constant practice is another key of success but maybe the most important aspect that needs to be incorporated into teaching methodology is addressing the gap between intuitive and everyday thinking and algorithmic approach which calls for additional changes in the curriculum.

Results of conducted research show that students have the biggest problems in understanding complex programming concepts that are abstract in nature and which are not intuitively clear. Research results also show that students have difficulties in understanding of programming tasks and in designing of appropriate algorithms for their solutions. By adding a new course that would include only moderately abstract concepts and structures programming novices would be able to focus on adapting to algorithmic way of reasoning rather than spending too much of their time on trying to grasp the complex syntax and concepts of programming. Development of such course and curriculum that would promote algorithmic way of thinking as well as testing of its effectiveness and efficiency will be a part of future research.

## 7. References

Alaoutinen, S. & Smolander, K. (2010). Student self-assessment in a programming course using bloom's revised taxonomy, *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education*, Laxer, C. (Ed.), pp. 155-159, ISBN 978-1-60558-820-9, Bilkent, Ankara, Turkey, June 26th - 30th, ACM, New York, NY, USA

Baldwin, L. P. & Kuljis, J. (2001). Learning programming using program visualization techniques. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, Sprague, R. H., Jr. (Ed.), pp. 1051-1058, ISBN 0-7695-0981-9, Maui, Hawaii, USA, January 3rd - 6th, IEEE, Washington, DC, USA

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, Vol. 30, No. 1, March 1998, pp. 257-261, ISSN 0-89791-994-7

Bennedsen, J. & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, Vol. 39, No. 2, June 2007, pp. 32-36, ISSN 0097-8418

Bergin, S. & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. *Proceedings of the 17th Annual Workshop on the Psychology of Programming Interest Group*, Douce, C. (Ed.), pp. 293-304, University of Sussex, Brighton, UK, June 28th - July 1st, Psychology of Programming Interest Group, UK

Dolinay, J.; Dostalek, P. & Vasek, V. (2010). Simple operating system RTMON for HC08 microcontrollers. *Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium*, Katalinic, B. (Ed.), pp. 0515-0516, ISSN 1726-9679, ISBN 978-3-901509-73-5, Zadar, Croatia, October 20th - 23rd, DAAAM International, Vienna, Austria, EU

Dolinay, J.; Dostalek, P. & Vasek, V. (2011). Graphical user interface simulators for lessons of real-time programming. *Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium*, Katalinic, B. (Ed.), Vol. 22, No. 1, pp. 395-396, ISSN 1726-9679, ISBN 978-3-901509-83-4, Vienna, Austria, November 23rd - 26th, DAAAM International, Vienna, Austria, EU

Fincher, S.; Robins, A.; Baker, B.; Box, I.; Cutts, Q.; de Raadt, M.; Haden, P.; Hamer, J.; Hamilton, M.; Lister, R.; Petre, M.; Sutton, K.; Tolhurst, D. & Tutty, J. (2006). Predictors of success in a first programming course. *Proceedings of the 8th Australasian Conference on Computing Education*, Tolhurst, D. & Mann, S. (Eds.), Vol. 52, pp. 189-196, ISBN 1-920682-34-1, Hobart, Australia, January 16th - 19th, ACS, Darlinghurst, Australia

Ford, M. & Venema, S. (2010). Assessing the success of an introductory programming course. *Journal of Information Technology Education: Research*, Vol. 9, No. 1, January 2010, pp. 133-145, ISSN 1539-3585

Gomes, A. & Mendes, A. J. (2007). An environment to improve programming education. *Proceedings of the 2007 International Conference on Computer Systems and Technologies*, Rachev, B.; Smrikarov, A. & Dimov, D. (Eds.), Art. No. 88, pp. 1-6, ISBN 978-954-9641-50-9, Rousse, Bulgaria, June 14th - 15th, ACM, New York, USA

Hanks, B.; McDowell, C.; Draper, D. & Krnjajic, M. (2004). Program quality with pair programming in CS1. *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, Boyle, R. (Ed.), pp. 176-180, ISBN 1-58113-836-9, Leeds, United Kingdom, June 28th - 30th, ACM New York, NY, USA

Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education*, Vol. 54, No. 4, May 2010, pp. 1127-1136, ISSN 0360-1315

Hu, M. (2004). Teaching novices programming with core language and dynamic visualization. *Proceedings of the 17th NACCQ*, Mann, S. & Clear, T. (Eds.), pp. 94-103, ISBN 0-476-00726-7, Christchurch, New Zealand, July 6th - 9th, NACCQ, Hillcrest, Hamiliton, New Zealand

Jenkins, T. (2002). On the difficulty of learning to program. *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, pp. 53-58, ISBN 0-9541927-1-0, Loughborough, UK, August 27th - 29th, ICS Subject Centre, Ulster, Ireland

Kinnunen, P. & Malmi, L. (2006). Why students drop out CS1 course? *Proceedings of the 2nd International Workshop on Computing Education Research*, Anderson, R.; Fincher, S. A. & Guzdial, M. (Eds.), pp. 97-108, ISBN 1-59593-494-4, University of Kent, Canterbury, UK, September 9th - 10th, ACM, New York, NY, USA

Lister, R.; Adams, E. S.; Fitzgerald, S.; Fone, W.; Hamer, J.; Lindholm, M.; McCartney, R.; Moström, J. E.; Sanders, K.; Seppälä, O.; Simon, B. & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, Vol. 36, No. 4, December 2004, pp. 119-150, ISSN 0097-8418

McCracken, M.; Almstrum, V.; Diaz, D.; Guzdial, M.; Hagan, D.; Kolikant, Y. B.-D.; Laxer, C.; Thomas, L.; Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, Vol. 33, No. 4, December 2001, pp. 125-140, ISSN 0097-8418

Nikula, U.; Gotel, O. & Kasurinen, J. (2011). A motivation guided holistic rehabilitation of the first programming course. *ACM Transactions on Computing Education (TOCE)*, Vol. 11, No. 4, November 2011, Art. No. 24, ISSN 1946-6226

Norman, D. A. & Draper, S. W. (1986). *User-Centered System Design: New Perspectives on Human-Computer Interaction*, Norman, D. A. & Draper, S. W. (Eds.), Lawrence Erlbaum Associates, ISBN 0-89859-781-1, Hillsdale, New Jersey

Pea, R. D. & Kurland, D. M. (1983). On the Cognitive Prerequisites of Learning Computer Programming. *Technical Report No. 18*, June 1983, Bank Street College of Education, Center for Children and Technology, New York, NY, USA

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, Vol. 2, No. 1, pp. 25-36, Baywood Publishing Co., Inc., New York, NY, USA

Pears, A.; Seidman, S.; Malmi, L.; Mannila, L.; Adams, E.; Bennedsen, J.; Devlin, M. & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, Vol. 39, No. 4, December 2007, pp. 204-223, ISSN 0097-8418

Peng, W. (2010). Practice and experience in the application of problem-based learning in computer programming course. *Proceedings of the International Conference on Educational and Information Technology (ICEIT)*, Yuting, L. (Ed.), Vol. 1, pp. 170-172, ISBN 978-1-4244-8033-3, Chongqing, China, September17th - 19th, IEEE, Piscataway, New York, NY, USA

Robins, A.; Rountree, J. & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Journal of Computer Science Education*, Vol. 13, No. 2, April 2003, pp. 137-172, ISSN 0899-3408

Smith, D. C.; Cypher, A. & Tesler, L. (2000). Programming by example: novice programming comes of age. *Communications of the ACM*, Vol. 43, No. 3, March 2000, pp. 75-81, ISSN 0001-0782

Smith, P. A. & Webb, G. I. (2000). The efficacy of a low-level program visualization tool for teaching programming concepts to novice C programmers. *Journal of Educational Computing Research*, Vol. 22, No. 2, April 2000, pp. 187-216, ISSN 0735-6331

Sorva, J.; Karavirta, V. & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, Vol. 13, No. 4, November 2013, Art. No. 15, ISSN 1946-6226

Tan, P. H.; Ting, C. Y. & Ling, S. W. (2009). Learning difficulties in programming courses: undergraduates' perspective and perception. *Proceedings of the IEEE International Conference on Computer Technology and Development*, Jusoff, K.; Othman M. & Xie Y. (Eds.), Vol. 1, pp. 42-46, ISBN 978-0-7695-3892-1, Kota Kinabalu, Malaysia, November 13th - 15th, IEEE, Los Alamitos, California,USA.

Wiedenbeck, S. (1999). Novice comprehension of small programs written in the procedural style. *International Journal Human-Computer Studies*, Vol. 51, No. 1, July 1999, pp. 71–87, ISSN 1071-5819

Wiedenbeck, S.; Ramalingam, V.; Sarasamma, S. & Corritore, C. L. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting With Computers*, Vol. 11, No. 3, January 1999, pp. 255-282, ISSN 0953-5438

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads*, Vol. 2, No. 4, December 2011, pp. 71-76, ISSN 2153-2184