Main problems of programming novices and the right course of action

Mario Konecki
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
mario.konecki@foi.hr

Marko Petrlić
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
marpetrli@foi.hr

Abstract. *Programming is required in all business systems and that makes programming courses vitally important. Regardless of this fact, programming courses experience rather high failure rates. There is a need to find the reasons for this kind of state and to determine the methods that can be used in order to improve mentioned situation. This paper gives an overview of conducted research regarding described state and provides a conclusion about the difficulty of programming courses and problems that can be detected. The paper also analyses existing efforts in improving education process and gives conclusion about possible courses of action.*

Keywords. Programming, novices, problems, education, learning

1 Introduction

Modern business would not be imaginable without computers and computer programs. Constant growth and business changes ask for quicker and more rapid development of computer programs which have become a vital part of almost all social systems. The importance of programming professionals in this kind of environment cannot be overstated and this puts a lot of responsibility on today's educational systems.

Nevertheless, teachers and faculties are struggling with many problems in teaching programming novices how to program and this consequently results in rather high failure and dropout rates. From the very beginning of programming profession there have been efforts to find the approach that would benefit programming novices and that would improve the quality of acquired knowledge and skills. Even though all these efforts have been made it is obvious that the problems of programming education are still present and that they are reoccurring in every new generation of programming novices.

General opinion that is widespread and accepted is that programming is hard to learn [2; 5; 9; 11; 14; 24; 25]. Every teacher can confirm that a large number of programming novices have problems learning even the most simple programming concepts. The question that needs to be answered is whether the programming is really so difficult and whether the main problem lies with programming novices or with existing teaching methods. Current state and methodology as well as programming novices' attitudes and habits need to be analyzed in order to determine the main sources of problems as well as potential steps for improvement.

This paper gives and overview of research made in this field as well as analysis of existing efforts that have been made in order to improve programming courses. This paper tries to give answers to questions about difficulty of programming courses, about problems that are mostly reoccurring and about measures and changes that can be undertaken in order to make programming courses closer to and easier for programming novices. Conclusion about the most prominent problems and the right course of action regarding improvement of programming courses as well as pointers for further research are also presented and discussed.

2 Programming courses' failure and dropout rates

The first question that needs to be answered is whether programming courses really have low success rate and how much programming novices have really learned after passing these courses. The point of view that is commonly accepted is that failure rates in introductory programming courses and dropout rates are rather high [19; 33]. Bennedsen and Caspersen [4] report average passing rate of 33% but they also report mixed results depending upon the size of class and other factors. They also note that there is a definite negative trend regarding the number of students that are interested in pursuing computer science studies.

Research has shown that students who are at the end of their studies frequently aren't able to produce a usable programming code and that less than 1% of programming novices continue to deal with programming after finishing their programming course [2]. It is also reported that very frequently students don’t know how to construct usable programs even at the end of their introductory programming courses [16]. This fact is usually explained by students’ lack of ability to perform program-solving tasks [16]. Lister et al. [16] also dealt with alternative explanation which claims that many students have little understanding of basic programming principles and that they lack the ability to systematically perform routine programming tasks. Lister et al. [16] conducted a research on students from seven countries which was performed in two phases that tested different aspects: the students’ ability to predict the output of small pieces of code and their ability to complete small amounts of missing code from short programming examples. The conclusion was that students lack skills that are needed for problem-solving activities since they were weak in both, especially second phase.

McCracken et al. [17] also report on alarming results which have shown that students aren’t able to program at the end of their introductory programming courses. They conducted research on 216 students and given the developed evaluation criteria the students had in average only 22.89 out of 110 points. As a part of their research they developed an assessment framework which includes the following learning objectives that the first year students of computer science are expected to learn which are also defined as needed steps for performing problem-solving activities:

1. Abstract the problem from its description
2. Generate sub-problems
3. Transform sub-problems into sub-solutions
4. Re-compose the sub-solutions into a working program
5. Evaluate and iterate

The developed assessment framework is aimed at analysis of success rate of introductory programming courses. Assessment methods used to evaluate success and failure rates of programming courses are also mentioned by other authors. Alaoutinen and Smolander [1] conducted a survey based on Bloom’s Revised Taxonomy with 87 students participating in the research in order to find whether the self-assessment tool can help students in recognizing the level of their programming knowledge and skills. The participating students consisted of both freshmen and advanced students. The survey gave 44 valid answers. The results have shown that in general students are quite good and precise in denoting of their programming knowledge and skills but it has also shown that many freshmen gave either overestimated or underestimated assessment of their knowledge and skills in programming. This fact can probably be explained by lack of experience in freshmen population regarding programming. Ford and Venema [8] conducted the research on 111 students using the Dehandi test [6]. They state that the number of correct answers of Dehandi test is relevant and can be used for assessing the performance of students on finished programming course and for detecting problems.

In order to be able to program, programming novices must comprehend three types of programming knowledge: syntactic, conceptual and strategic [2]. Syntactic knowledge refers to understanding of means of expression in some particular programming language and can be compared to knowing words and grammatical rules of any spoken language. Conceptual knowledge refers to constructs and principles of programming that require programming novices to develop proper mental models. Strategic knowledge refers to skills of understanding and structuring the problem and solving the problem by constructing a usable algorithmic solution. Programming novices often have problems in understanding the way of thinking that is needed to instruct computer to perform some desired task.

Don Norman stated that the gap between novices’ 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 [20]. 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 [20]. Educational process is trying to bring user closer to the system by teaching students about the system, but efforts to bring programming closer to the user have also been made and are presented later in this paper, although none of these developed approaches, methods and tools have become widely accepted in formal education.

3 Reasons of problems in programming courses

The second question that needs to be answered is the question about the reasons of low success rates in introductory programming. Pea and Kurland [21] state that all programming novices possess some intuitive idea about programming and its concepts, however Ben-Ari [3] claims that students do not possess an effective model of how computer works. He also states that this model becomes more accurate as students are introduced to more and more computer technology but it still cannot be concluded wheatear this is a good starting point or a source of problems. This question is also supported by already established gap in the way students intuitively think and algorithmic and problem-solving way of thinking which is needed to instruct computers [20].

Hawi [12] conducted a research on 45 undergraduate students that have finished their programming courses in order to obtain their perception about casual attributions of success and failure regarding programming courses. He used narrative interview to get more accurate results. He obtained 10 casual attributions: learning strategy, lack of study, lack of practice, subject difficulty, lack of effort, appropriate teaching method, exam anxiety, cheating, lack of time, and unfair treatment. Learning strategy was on the top of the list since 40% of the participants mentioned it as one of causal attributions.

Another research was conducted by Kinnunen and Malmi [15] who wanted to find out the programming courses dropout reasons. Their research was conducted on 105 students that were given the questionnaire and 18 of these students were additionally interviewed to get more accurate results. The results indicate multiple reasons of dropout with the lack of time and the lack of motivation as the most frequent reasons. Both of these reasons were also affected by various factors: perceived difficulty of the course, general difficulties with time managing and planning studies, or the decision to prefer something else, etc. Their research shows large amount of factors that need to be addressed in order to reduce dropout rates and they state that simple actions of teachers in order to improve the course success rate may be ineffective.

Gomes and Mendes [9] state that students have problems in programming and creating algorithms because they don’t know how to solve problems in proper way. This promotes the idea that these problem-solving skills are the place to start with potential solution. Although introductory programming course should train students in solving problems by usage of some chosen programming language, students have many problems in using these skills to the point of having problems with even the most basic concepts.

Gomes and Mendes [9] state that these problems could have many different reasons:

- Programming demands a high abstraction level.
- Programming needs a good level of both knowledge and practical problem solving techniques.
- Programming requires a very practical and intensive study, which is quite different from what is required in many other courses (more based in theoretical knowledge, implying extensive reading and some memorization).
- Usually teaching cannot be individualized, due to common classes’ size.
- Programming is mostly dynamic, but usually thought using static materials.
- Teachers’ methodologies many times don’t take into consideration the student’s learning styles. Different students have different learning styles and can have several preferences in the way they learn.
- Programming languages have a very complex syntax with characteristics defined for professional use and not with pedagogical motivations.

Nikula et al. [19] conducted a five-year research in which they tried to answer the question about reasons of low pass rates in introductory programming courses. They recorded pass rates each year along with the data about the deliverables and attending rate. Each year they conducted four surveys: initial, midterm, final, and dropout. Initial survey dealt with demographic data and initial skills, midterm survey dealt with progress that has been made, final survey dealt with success of students and their perception of finished course, and dropout survey was given to those students who did not finish the course. During the whole period of research changes were made each year in order to improve the course. Nikula et al. [19] concluded that the low pass rate is affected mostly by three factors: programming as a discipline, course arrangements, and student behavior.

Since it was already mentioned that most authors agree that to learn how to program is hard and challenging task [2; 5; 9; 11; 14; 24; 25] it can be said that programming itself is a complex area to understand and master. Programming courses require a lot of concepts and technical details from their students and the pure arrangement of these courses makes them difficult for students, especially the ones who are encountering the programming for the first time. Student behavior is influenced by intrinsic and extrinsic motivation. Students that are intrinsically motivated and find rewarding to learn programming finish programming courses regardless of other factors. Students that are however extrinsically motivated and that are learning programming for example just because it is mandatory to do so, find themselves frustrated and impeded by various complexities that are part of programming courses. Nikula et al. [19] also state that various de-motivators, that lead to students being unsatisfied due to accumulated complexities that come with the programming course, are also important reason of high dropout rate.

Tenenberg and Fincher [30] conducted a study that included over 300 participants form 21 institutions in 4 countries. All participants were divided in two groups depending whether they are at the beginning or end of their studies. The study was aimed at exploring how well the students understand the software design process. They were given a decomposition task to find out about their analytical skills as well as about their ability to use design concepts to create proper solution structure. Students were also given a design criteria prioritization task where students were asked to choose criteria that they consider most and least important for some particular design scenario. Results have shown that some design behaviors are and some are not dependent on educational level. Recognizing ambiguity and use of standard design scenarios are related to educational level whereas design criteria evaluation is not. Information gathering and representation of interactions between elements appeared to be context-dependent and thus most amenable to instructional changes.

Wiedenbeck [31] reports on conducted research in which novice programmers were given short procedural and object-oriented examples with objective to determine their comprehension of given programs. The main question was whether programming novices' mental representation is more focused on domain-level or program-level knowledge and whether there is a difference in mental representation regarding procedural and object-oriented programs. The results have shown that programming novices' function-related comprehension is more prominent in object-oriented programs and that data flow and program-related comprehension was less prominent whereas program-related comprehension was more prominent in procedural programs indicating that there are certain aspects of object-oriented programming that are clearly harder to understand because of more complex concepts which lead to weaker program-oriented comprehension.

Another research related to one previously described was conducted by Wiedenbeck et al. [32] that was aimed at determining the difference in mental representation and comprehension of programming novices regarding object-oriented and procedural programming. The research was conducted on 86 programming novices. Novices were given short and long programs to analyze and answer 12 comprehension question that were divided into 4 groups of 3 question each regarding different aspects: elementary operations, control flow, data flow, and function. For short programs there was no significant difference between object-oriented and procedural group regarding the number of correct answers, however object-oriented group has shown larger tendency towards questions about program function. This effect was lacking in analysis of large programs where procedural group was generally superior on all questions with overall score being 15% higher.

Some authors report that different selection of programming notation can support particular programming concepts [10; 31; 32] which could partially explain research results described by Wiedenbeck et al. [32]. Jenkins [14] comments on importance and existence of students' aptitude for programming. He claims that commonly mentioned problem solving skills and mathematical ability have no tests that would be conclusive and could measure programming aptitude in sufficiently satisfactory and convenient way and he also states that if this is so, the reasons of difficulty to learn how to program must be searched through more cognitive view of overall learning process. He states two cognitive factors as potential answers about difficulty to learn programming: learning style and motivation. Wrong state of either of factors mentioned would result in student not being able or willing to learn properly and thus not understanding programming in satisfactory way.

Pea [22] claims that certain bugs exist that are commonly found in understanding of programming and that are present in entire programming novices population. He claims that these bugs are reoccurring and that they are more related to the proper way of instructing computers than to design of some particular programming language. There are three classes of mentioned reoccurring bugs:

- Parallelism bug
- Intentionality bug
- Egocentrism bug

Parallelism bug refers to false understanding by which computers are able to deal with multiple programming lines simultaneously. This bug for example refers to situation in which the computer would be able to look back in program and execute some condition after the condition would be fulfilled later in program execution. Intentionality bug refers to students' presumptions about the function of program and about its output judging by only a small part of program code. Students are frequently reminded on some functionality by some portion of code and they conclude that it does something without looking at all programming instructions and without proper analysis of program flow. Egocentrism bug refers to attitude towards computers which results in lack of programming instructions because students think that computer will somehow be able to fill the gaps itself and perform the job they want. Important and vital instructions are frequently omitted and programming code isn't precise enough to do the job as expected.

When considering problems relevant to programming courses it is important to think about the reasons why students are enrolling these courses. One research reports that only 22% of students in the first year of study studied programming because of interest in this area, 40% because of career reasons and 35% just because it was mandatory to do so. 5% of students didn't participate in the research [5]. Bergin and Reilly [5] conducted study on 110 students of introductory programming course and they report that positive attitude toward programming is influenced by both intrinsic and extrinsic motivation with intrinsic motivation being more important and more effective in increasing programming performance. Some authors also report on various means aimed at increasing the motivation level, such as the use of web or game programming examples in order to make programming more interesting in comparison to standard programs that are commonly presented to students [5].

4 Efforts made to improve programming courses

The third question that needs to be answered is what efforts have been made and what can be done to improve the success rates of introductory programming courses. Nikula et al. [19] in their five-year research concluded that eliminating de-motivators that students encounter because of high complexity of programming course is the first step in answering the question about how can the pass rate be improved. Second step would be to increase the intrinsic motivation by including additional programming tools and making programming projects more interesting and useful from students' point of view. Finally, the third step would be to deal with extrinsic motivation which they addressed in a way that assignments from previous years were no longer accepted, students were required to finish at least 40% of their assignments on weekly basis and students were also required to submit initial versions of their projects three weeks before submitting the final versions. These measures have greatly increased the predictability of students behavior. Nikula et al. [19] finally conclude that in order to improve passing rate of programming course the following steps need to be taken:

1. Eliminate de-motivators
2. Increase intrinsic motivators in the course by making it more interesting and useful
3. Introduce extrinsic motivators to increase the predictability of students behavior

Jenkins [14] states several things that should change in order to improve the situation regarding the difficulty to learn programming:

- Programming should never be taught before the second year of any course
- The language used should be chosen for pedagogic suitability and not because it is popular in industry
- Programming should be taught by those who can teach programming and not those who can program
- Programming courses should be designed to be flexible to allow different students to learn in different ways
- There should be no summative (continuous) assessment to ease pressure on students
- Departments should acknowledge that programming is difficult and supply adequate support to students

When considering programming problems, the teaching methods must be taken into consideration and evaluated. Fincher et al. [7] report that deep approach to learning has positive impact on success of students in programming courses opposite to surface learning that is reported to have negative effect. However, Jenkins [14] states that since programming is a skill, not only knowledge, deep and surface learning are both needed simultaneously in order for students to perform well since deep learning gives expertise in particular segment of programming but surface learning is important to see the overall picture and to grasp all concepts and parts of programming skill.

Hu [13] reports results of conducted research among students of programming course which show that 90% of students prefer to learn theory and practice simultaneously when learning programming. He also promotes visualization as an efficient mean to improve students programming performance. Tan et al. [29] conducted a study on 182 undergraduate students who finished introductory programming course. The results of research have shown that students prefer learning by examples and practice over classic lectures that decrease their interest. Consequently, authors propose game-based learning framework as a better way of teaching in introductory programming.

Hanks et al. [11] compared performance of solo and paired students during two different semesters. They report that students who worked in pairs turned in more programs that compiled correctly. They state that students who worked in pairs wrote more functional programs although no claims can be stated regarding the quality of programs' design but paired students were found to be more confident and satisfied in their work. Nagappan et al. [18] also conducted an experiment to determine the effects of pair programming in introductory programming course. Results have shown that paired students were more self-sufficient, more active and that they achieved better results compared to solo students. They also conclude that paired programming can reduce students' frustrations regarding programming.

Gomes and Mendes [9] think that students' main problem is lack of problem-solving skills that would enable them to create valid algorithms so they created a system named SICAS (Interactive System for Construction of Algorithms and its Simulation) which promotes training in construction and testing of algorithms. This system is based on constructivist approach towards teaching problem-solving skills by doing and trying and in that way students are able to create their own knowledge. Their point of view is supported by Don Norman's definition of gap between students' intuition and problem-solving skills that are needed to instruct computers [20].

Yadin [33] reports on determined factors of success in introductory programming courses that helped to reduce failing number of students by over 77%. He states three such factors:

1. Usage of Python to reduce complexity of syntax thus allowing the students to focus on

problem-solving skills and creation of algorithms

2. Usage of visualization tool to help clarify abstract and complex concepts
3. Individual assignments for every student that are aimed at development of stronger learning habits

Sorva et al. [28] conducted a research to evaluate the usefulness of program visualization tools in introductory programming. They state that there are indications that these kind of tools are beneficial, however they also state that research to date is not conclusive enough for drawing conclusions that consider students' engagement. Smith and Webb [27] developed their own visualization tool to try to clarify abstract concepts to programming novices. Their research has shown that students indeed do benefit from such tools and that they were able to grasp new concepts quicker and better compared to students that were taught in traditional way.

Smith et al. [26] report on creating the new approach towards teaching novices programming that addresses the fact that there hasn't been any approach that would be widely accepted since 1960s when intensive research in this area stared. They created a tool called Stagecast Creator which is based on two technologies: programming by demonstration (PBD) and visual before-after rules. The results of using Stagecast Creator have shown that it is well suited for novices, even children who are able to create programs by demonstrating to computer what to do in graphical way rather than to program in textual code. Stagecast Creator is based on visual before-after rules where the user has to define only start and end state in order to describe desired visual simulation that is the domain to which this tool is limited.

Baldwin and Kuljis [2] report on using visual metamorphs and analogies in order to try to bring programming concepts closer to the user. They report on a series of visual languages and systems that are aimed at teaching programming without complicated code and abstract concepts or difficult programming syntax by using visual metamorphs and analogies. However, they state that there is not enough evidence to fully conclude that visual languages are easier and more beneficial than non-visual ones although they believe that visual nature of this languages as well as their reduced complexity must somehow be better that non-visual languages.

5 Conclusion

Programming professionals are of vital importance for all aspects of business and educational systems have a large responsibility to produce this kind of experts. However, the problems in programming education are persisting and reoccurring in every generation of programming novices. The fact that programming courses have high failure and dropout rates has become a general opinion [19; 33]. Bennedsen and Caspersen [4] report on low passing rate of 33% and the fact that less and less students are willing to study computer science. This negative trend is also reported by Baldwin and Kuljis [2]. It has been determined that many students don't know how to write computer programs at the end of their studies [2] but also that they aren't able to produce valid programs even at the end of their introductory programming courses [16]. The same results have been reported by McCracken et al. [17] who conducted a research in which the average score of students was only 22.89 out of 110 points.

Don Norman states that there is a huge gap between the way programming novices are used to think and problem-solving way of thinking required to program [20] and this view is also supported by other authors [16]. It can be concluded that programming courses indeed have rather high failure and dropout rates and that many students have difficulties in passing these courses. The question that however still remains unclear is whether the source of these problems is mostly programming novices or educational methods related.

Pea and Kurland [21] state that all programming novices have some sort of idea about programming concepts, but Ben-Ari [3] claims that these models are mostly ineffective and a potential source of problems. This corresponds to gap mentioned by Don Norman [20] and draws a focus on teaching and training novices in algorithmic thinking and problem-solving skills. Hawi [12] reported on 10 determined casual attributions of failure in introductory programming with learning strategy as the most prominent attribution. Kinnunen and Malmi [15] promote the view that multiple actions must be undertaken in order to address dropout reasons which are multiple with the lack of time and lack of motivation as the most frequent. Nikula et al. [19] state intrinsic motivation as important factor in passing programming courses that makes students perform better.

Reported results of one conducted research [5] shows that only 22% of first-year students decided to study programming because of personal interest. This supports the research that has been made regarding importance of intrinsic motivation [19] that could be lacking if students are not enrolling programming courses or studies because of their interest but because of some external factors.

All stated facts show that the reasons of low passing rates in introductory programming are still not completely clear since different researches report different factors of success or failure. However, it is clear that it is a complex question and that multiple factors are involved. More research on these factors need to be conducted in order to determine the key factors that are producing rather low passing rates in introductory programming courses.

Different authors also report different measures aimed at improving the situation in programming courses. Nikula et al. [19] propose eliminating de-motivators and frustration factors and increasing students' intrinsic motivation by more interesting and useful examples. Jenkins [14] states that programming shouldn't be taught in the first year of study and that the teachers' pedagogical and teaching skills are of great importance for success as well as selection of proper and suitable programming language. Yadin [33] reports on failing reduced by over 77% by using Python that reduces syntax complexity, by usage of visualization tool and by giving students individual assignments. Sorva et al. [28] and Smith and Webb [27] have reported on positive results of using visualization tools for teaching abstract programming concepts. Baldwin and Kuljis [2] also report on using visual metamorphs and analogies as beneficial means for making abstract concepts more clear.

Hu [13] reports that 90% of students in his research preferred practice along with theory and Tan et al. [29] reports on students' preference to learn by example over theoretical lectures that lower their interest in course. This point of view is also supported by Jenkins [14] who states that programming is a skill, not only knowledge and that deep and surface learning should be included simultaneously in order for students to grasp overall picture as well as fine implementation details. Another approach towards improving programming courses' results is reported by Hanks et al. [11] and Nagappan et al. [18] who report on benefits of paired over solo programming.

When looking at the research about reasons of high failure rates, there are many aspects and means that are proposed in different researches. It can be concluded that the methods for improving results of programming courses are multiple, with each of them being more or less successful in some particular context. None of them is however widely spread and accepted in formal education or on a larger scale. Therefore, more research is needed in order to classify reasons and efforts for improvement and to determine the most efficient methods for reducing failing and dropout rates in introductory programming courses.

References

[1] Alaoutinen, S; Smolander, K. Student self-assessment in a programming course using bloom's revised taxonomy. *In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 155-159), ACM, New York, NY, USA, 2010.

[2] Baldwin, L. P; Kuljis, J. Learning programming using program visualization techniques. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, pages 1051-1058, IEEE, Washington, DC, USA, 2001.

[3] Ben-Ari, M. Constructivism in computer science education. *ACM SIGCSE Bulletin*, 30(1):257-261, 1998.

[4] Bennedsen, J; Caspersen, M. E. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32-36, 2007.

[5] Bergin, S; Reilly, R. The influence of motivation and comfort-level on learning to program. In *Proceedings of the 17th Annual Workshop on the Psychology of Programming Interest Group*, pages 293-304, University of Sussex, Brighton, UK, 2005.

[6] Dehnadi, S. Testing programming aptitude. In *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, pages 22-37, Brighton, UK, 2006.

[7] 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. Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Conference on Computing Education*, pages 52:189-196, ACS, Darlinghurst, Australia, 2006.

[8] Ford, M; Venema, S. Assessing the success of an introductory programming course. *Journal of Information Technology Education: Research*, 9(1):133-145, 2010.

[9] Gomes, A; Mendes, A. J. An environment to improve programming education. In *Proceedings of the 2007 international conference on Computer systems and technologies*, pages 88:1-6, ACM, New York, USA, 2007.

[10] Good, J; et al. Novices and Program Comprehension: Does Language Make a Difference? In *Proceedings of 19th Annual Conference of the Cognitive Science Society*, pages 936–937, Stanford University, 1997.

[11] Hanks, B; McDowell, C; Draper, D; Krnjajic, M. Program quality with pair programming in CS1. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 176-180, Leeds, United Kingdom, 2004.

[12] Hawi, N. Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education*, 54(4):1127-1136, 2010.

[13] Hu, M. Teaching novices programming with core language and dynamic visualization. In *Proceedings of the 17th NACCQ*, pages 94-103, Christchurch, New Zealand, 2004.

[14] Jenkins, T. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, pages 53-58, Loughborough, UK, 2002.

[15] Kinnunen, P; Malmi, L. Why students drop out CS1 course? In *Proceedings of the 2006 International Workshop on Computing Education Research*, pages 97-108, ACM, 2006.

[16] 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. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150, 2004.

[17] McCracken, M; Almstrum, V; Diaz, D; Guzdial, M; Hagan, D; Kolikant, Y. B.-D; Laxer, C; Thomas, L; Utting, I; Wilusz, T. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4):125–140, 2001.

[18] Nagappan, N; Williams, L; Ferzli, M; Wiebe, E; Yang, K; Miller, C; Balik, S. Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin*, 35(1):359-362, 2003.

[19] Nikula, U; Gotel, O; Kasurinen, J. A motivation guided holistic rehabilitation of the first programming course. *ACM Transactions on Computing Education (TOCE)*, 11(4), 24, 2011.

[20] Norman, D. A; Draper, S. W. *User-Centered System Design: New Perspectives on Human-Computer Interaction*, Erlbaum, Hillsdale, NJ, 1986.

[21] Pea, R. D; Kurland, D. M. On the Cognitive Prerequisites of Learning Computer Programming. *Technical Report No. 18*, Bank Street College of Education, New York, NY. 1984.

[22] Pea, R. D. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1):25-36, 1986.

[23] Pears, A; Seidman, S; Malmi, L; Mannila, L; Adams, E; Bennedsen, J; Devlin, M; Paterson, J. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, 39(4):204-223, 2007.

[24] Peng, Wu. Practice and experience in the application of problem-based learning in computer programming course. *In Proceedings of the International Conference on Educational and Information Technology (ICEIT)*, pages 1:170-172, IEEE, 2010.

[25] Robins, A; Rountree, J; Rountree, N. Learning and Teaching Programming: A Review and Discussion. *Journal of Computer Science Education*, 13(2):137-172, 2003.

[26] Smith, D. C; Cypher, A; Tesler, L. Programming by example: novice programming comes of age. *Communications of the ACM*, 43(3):75-81, 2000.

[27] Smith, P. A; Webb, G. I. The efficacy of a low-level program visualization tool for teaching programming concepts to novice C programmers. *Journal of Educational Computing Research*, 22(2), 187-216, 2000.

[28] Sorva, J; Karavirta, V; Malmi, L. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, 13(4), 15, 2013.

[29] Tan, P. H; Ting, C. Y; Ling, S. W. Learning difficulties in programming courses: undergraduates' perspective and perception. In *Proceedings of the IEEE International Conference on Computer Technology and Development*, pages 1:42-46, IEEE, Kota Kinabalu, Malaysia, 2009.

[30] Tenenberg, J; Fincher, S; et al. Students designing software: a multi-national, multi-institutional study. *Informatics in Education*, 4(1):143–162, 2005.

[31] Wiedenbeck, S. Novice comprehension of small programs written in the procedural style. *International Journal Human-Computer Studies*, 51(1):71–87, 1999.

[32] Wiedenbeck, S; Ramalingam, V; Sarasamma, S; Corritore, C. L. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting With Computers*, 11(3):255-282, 1999.

[33] Yadin, A. Reducing the dropout rate in an introductory programming course. *ACM Inroads*, 2(4):71-76, 2011.