Perceived Quality of Verificator in Teaching Programming

T. Orehovački, D. Radošević and M. Konecki
University of Zagreb, Faculty of Organization and Informatics, Varaždin, Croatia
{tihomir.orehovacki, danijel.radosevic, mladen.konecki}@foi.hr

Abstract – Verificator is an educational tool meant for learning syntax and semantics of the C++ programming language at the introductory programming courses. With an objective to explore quality of the Verificator from student's perspective, a pilot study was carried out. Data were collected by means of an online post-use questionnaire. The analysis of collected data uncovered pros and cons of the Verificator perceived by novice programmers. Empirical findings are presented and discussed.

I. INTRODUCTION

Drawing on our teaching experiences at different programming courses, we have created an initial version of an educational tool meant for writing code in C++ programming language which was named as Verificator [1]. The idea behind the development of Verificator emerged from observations of teachers. Students often tend to adopt some bad programming habits in their attempt to pass the exam. For instance, some students have learned even big fragments of program code by heart, almost without any understanding. There were also cases when students have written large amounts of program code without any syntax or logical testing. The set forth has resulted in a huge number of errors, discouraging students from learning programming. The use of Verificator prevents such bad practice and motivates students to adopt good programming habits.

The evaluation of the quality is an integral part of the life cycle of every software product. According to ISO/IEC 25000 [4], quality is a "degree to which a software product satisfies stated and implied needs when used under specified conditions". In the Software Product Quality Requirements and Evaluation (SQuaRE) series of the international standards [5][6] the quality evaluation framework consists of three models: the product quality model, the quality in use model, and the data quality model. Each of the aforementioned models offers a set of quality characteristics relevant to various groups of stakeholders.

This paper deals with the assessment of the perceived quality of Verificator from students' perspective. The remainder of the paper is structured as follows. Next section provides a brief overview of theories and models that form the background to our research. Features of the Verificator are described in the third section. Findings of the pilot study are presented in the fourth section. Conclusions and future research directions are contained in the last section.

II. LITERATURE REVIEW

A. Theoretical background

The constructs that were used and examined in our research are based on several models: Technology Acceptance Model (TAM), Information System Success Model (ISSM), Unified Theory of Acceptance and Use of Technology (UTAUT), Expectation Confirmation Theory (ECT) and Theory of Flow (TOF).

TAM is a theoretical framework which provides insight into the determinants of the adoption and use of the information technology. From its latest version [7], we adopted four different constructs. Perceived ease of use is the degree of ease associated with the use of the system. Perceived usefulness represents the extent to which an individual believes that using the system will enhance his or hers performance in assignment performance. Self-efficacy denotes the degree to which an individual believes that he or she has the ability to perform specific assignment. Relevance refers to individual's perception regarding the degree to which the target system is relevant for the task at hand.

ISSM was developed as a framework and model for measuring the complex-dependant variable in IS research. For our study, we adapted one construct from its ten-year update [8]. System quality is the extent to which a person perceives the quality of the system along with the process of using the system.

UTAUT represents a synthesis of eight technology acceptance studies. From its recent extension [9] we adapted one construct. Performance expectancy (in our study value-added) is the degree to which the use of the system will provide benefits to users.

ECT [10] examines cognitive beliefs and affects influencing one's intention to continue using information systems. Two constructs were adopted from this model. Confirmation is the degree of fulfilled expectation when individual was using the system. Satisfaction refers to a pleasurable or positive emotional state resulting from the appraisal of one's job.

TOF [14] is a model which examines mental state, involvement and enjoyment of a person in the process of some activity. One construct was adopted from this model. Flow denotes the extent to which an individual is fully immersed and enjoys in specific assignment.

Apart from the aforementioned constructs, we added three additional constructs that are specific for our

research. Helpfulness is the degree to which Verificator helps in better understanding of programming concepts. Good programming habits refer to the extent to which individual can utilize better his or hers programming skills and develop good programming habits when using Verificator. Feedback represents the degree to which system notifies users about the progress or status of the task at hand through appropriate messages.

B. *Related work*

There are many tools that were developed to help one's education process of computer programming. A brief overview of these tools is offered in [3]. Lane and VanLehn [18] found that tools developed for learning programming can positively influence on students' perception and motivation.

TAM is the most popular model for technology acceptance in the online learning domain [19]. There are a few studies that deal with constructs of TAM in the field of computer programming but they are not specific to a certain learning tool. Orehovački et al. [2] uncovered that perceived usefulness and perceived ease of use have influential role weather students will accept the learning tool and future behaviors towards its use. According to Ramalingam et al. [13], self-efficacy and mental models significantly affect the process of learning programming. Webster et al. [20] emphasize that flow plays a significant role in human-computer interactions. Results of the study conducted by Potosky [17] showed that there is a link between flow and post-training programming efficacy. Previous programming experience can also have significant role in programming education and acceptance of supporting learning tools [12][16]. Finally, findings of several studies indicate that computer anxiety correlates with the students prediction of their final grade and with perception of their own programming skills and habits [11][15].

III. VERIFICATOR

Verificator[1] is an educational tool developed with the aim to improve the process of teaching programming at the university beginner's level thus helping students to acquire programming skills more easily [1]. The user interface of the Verificator is displayed in Figure 1.

After starting with some basic functionalities like preventing copy/paste of program code and obligatory syntax checks within given intervals, over years, Verificator has been enriched with several novel features such as program testing mode, tutor, time control, program structure analysis, etc. The aforementioned functionalities will be explained in more detail in following sub-sections.

A. *Personalization of programs*

Students are asked to fill out the welcome form with some general information about themselves and the assignment (e.g. name, id, program name and description). These data are included into the program code in the form of comments together with a checksum by means of which it can be examined whether the program code was completely written in Verificator or not.

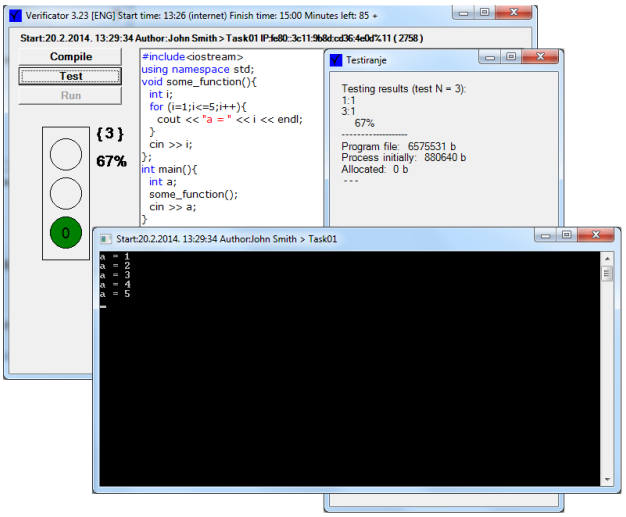

Figure 1. User interface of Verificator

B. *Time limitation for solving the assignment*

Students should solve the assignment within a predefined time interval. To avoid the problem with different system time at different computers, the time is controlled with the use of time servers on the Internet. After the time runs out, a student can still run the program, but he or she cannot modify it.

C. *Prevention of copying programs from colleagues*

Verificator prevents copying program code from external sources. In that respect, the overall code, apart from the given library, must be entered by the student. This feature is connected with obligatory intervals of syntax and logical program testing. Therefore, it is not possible to retype the program code. Instead, it has to be verified within the specified intervals.

D. *Adoption of good programming habits*

Findings of prior studies [1][3] indicate that students who employ standard programming environments to some extent tend to adopt bad programming habits such as learning programming code by heart, programming without syntax and logical checks, etc. Verificator requires from students to check the syntax of their programs after every 10 new lines of code which is graphically illustrated with traffic lights (red light indicates that is no longer possible to compile the program). Furthermore, program has to be checked logically within the given intervals (expressed in the number of program blocks) in a way that all program blocks have to be reached during the execution.

E. *Help for easier detection of syntax and logic errors*

Apart from the standard errors and warnings provided by compiler, Verificator offers further explanation of possible causes of these errors (e.g. unmatched curly brackets or parentheses, incorrect use of certain operators, etc.). In addition, Verificator also contains a tool for the analysis of the program structure as well as the debugging tool.

[1] http://gpml.foi.hr/teaching_tools.html

IV. PILOT STUDY

A. Procedure and Apparatus

During the winter semester of the academic year 2013/2014 the study participants employed Verificator to solve fifteen diverse lab-based programming assignments. Students also had the opportunity to use Verificator in order to prepare themselves for the lab-based exercises which took place once a week. At the end of the semester, students were asked to complete the post-use questionnaire consisted of 7 items related to their demographic characteristics as well as 50 items meant for evaluating twelve different facets of the perceived quality of Verificator in the context of teaching programming. The responses to the items referring to specific aspects of the perceived quality were modulated on a five-point Likert scale (1 – strongly agree, 5 – strongly disagree). Items marked with asterisk are reverse coded.

B. Participants

A total of 121 respondents ranging in age from 19 to 28 years (M = 20.89, SD = 1.315) took part in the study. The sample was composed of 83.47% male and 16.53% female second-year undergraduate students enrolled in the Information Systems study programme. The majority of students (43.80%) perceived their programming skills as good. Similarly, most of the respondents (56.20%) had good knowledge of C++ programming language syntax. Considering the frequency of using the Verificator, 83.47% of students employed it between once and twice a week while 63.64% of them used the aforementioned tool between one and three hours per week. Finally, the majority of participants (84.30%) had up to four years of programming experience.

C. Findings

According to the data presented in Figure 2, majority of pilot study subjects (61.49%) believe that the employment of the Verificator contributes to the development of good programming habits. For instance, 63.64% of students reported that Verificator motivates them to be more disciplined while solving the problem-based assignments (M = 2.40, SD = 1.158). In addition, 59.50% of participants stated that Verificator encourages them to write their program solutions more clearly (M = 2.62, SD = 1.349).

Results of data analysis displayed in Figure 3 indicate that 54.88% of students perceive Verificator as a helpful tool in the context of teaching programming. While 63.64% of respondents reported that Verificator is helping them in the interpretation of syntax errors (M = 2.46, SD = 1.133), merely 44.63% of them believe that Verificator explains messages provided by the interpreter in a more intuitive manner (M = 2.74, SD = 1.061).

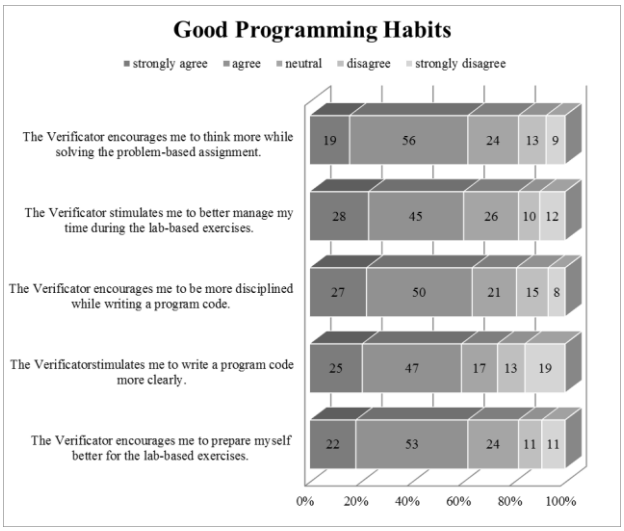

Figure 2. Perceived effect of using the Verificator on development of good programming habits

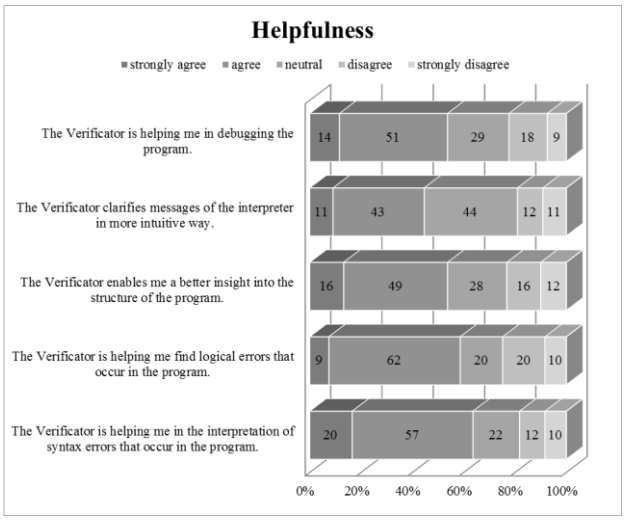

Figure 3. Perceived helpfulness of Verificator

As can be observed from the responses to questionnaire items shown in Figure 4, 64.30% of study participants think that messages provided by the Verificator are of high quality. Namely, majority of them (73.55%, 62.81%, 62.81%, 62.81%, and 59.80%, respectively) agree that messages displayed by Verificator are useful (M = 2.17, SD = 0.898), understandable (M = 2.42, SD = 0.955), accurate (M = 2.30, SD = 0.823), trustworthy (M = 2.28, SD = 0.839), and clear (M = 2.45, SD = 0.939).

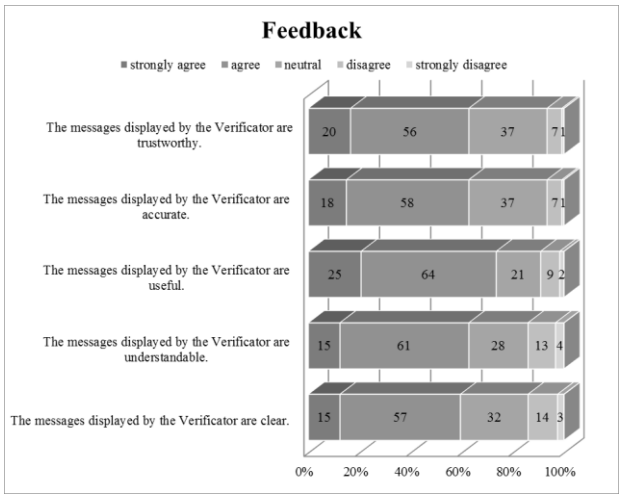

Figure 4. Perceived quality of messages provided by Verificator

The data presented in Figure 5 imply that 46.28% of study participants believe that Verificator has sufficient level of system quality. Namely, 69.42% of the respondents are pleased with the responsiveness of Verificator (M = 2.32, SD = 0.896). On the other hand, only 29.75% of them agree that Verificator is stable and reliable in its operation (M = 3.25. SD = 1.113).

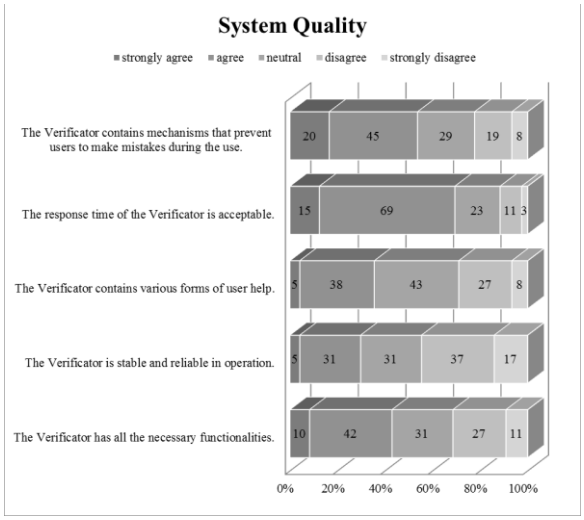

Figure 5. Perceived system quality of Verificator

The findings presented in Figure 6 indicate that Verificator made an impression on only one third (33.33%) of students. More specifically, 40.50% of participants reported that interaction with Verificator was a frustrating experience for them (M = 2.93, SD = 1.209). On contrary, 31.40% of study subjects were satisfied with using the Verificator (M = 3.13, SD = 1.072).

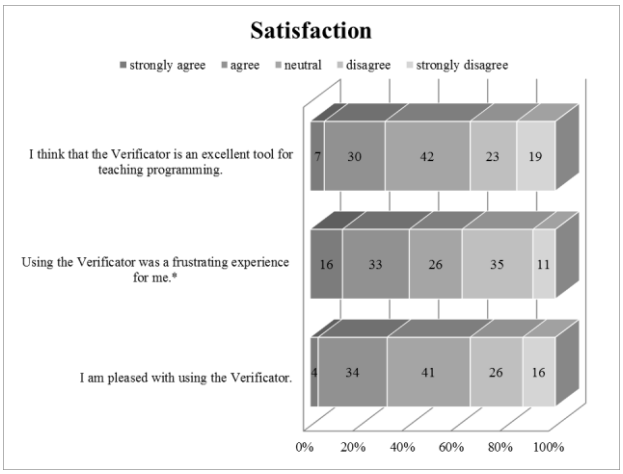

Figure 6. Satisfaction with using the Verificator

According to the results of data analysis displayed in Figure 7, only 41.05% of students agree that the interaction with the Verificator was as they expected. While 48.76% of participants reported that majority of their expectations on interaction with the Verificator were confirmed (M = 2.68, SD = 1.035), 34.71% of them stated that the use of the Verificator has not met their expectations (M = 3.03, SD = 1.016).

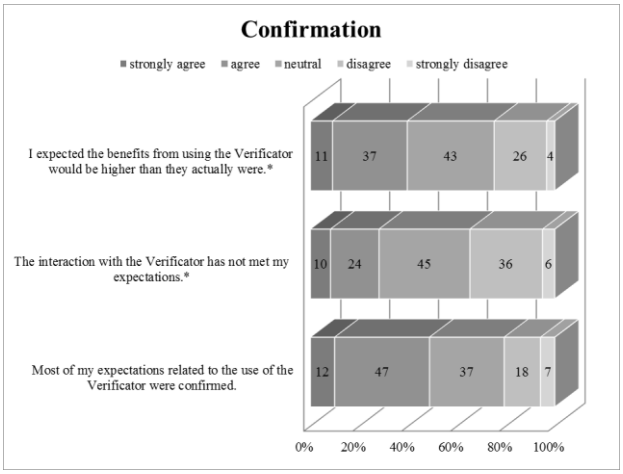

Figure 7. Confirmation of expectations related to the interaction with Verificator

As can be observed from the responses to questionnaire items shown in Figure 8, 30.74% of study participants believe that frequent employment of Verificator improves their programming skills. For instance, 45.45% of students think that there is a positive correlation between frequency of using the Verificator and number of points obtained during lab-based sessions (M = 2.93, SD = 1.219). On contrary, 36.36% of participants reported that frequency of using the Verificator has nothing with their efficiency in learning programming concepts (M = 2.78, SD = 1.092).

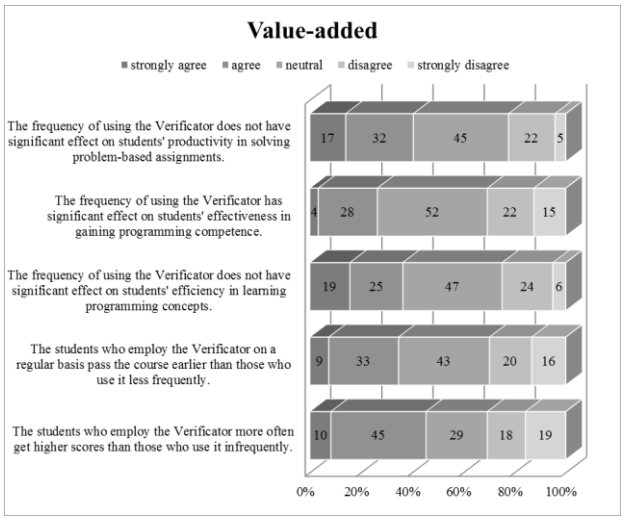

Figure 8. Value-added of the Verificator

Considering the data presented in Figure 9, Verificator is a relevant tool for teaching programming. Namely, 46.28% of students think that Verificator is convenient for learning programming concepts (M = 2.85, SD = 1.188).

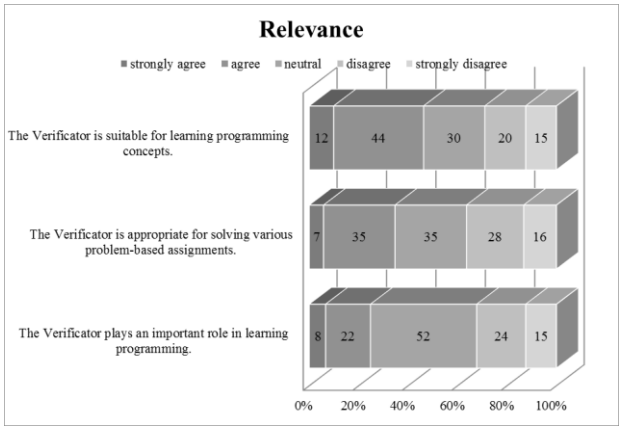

Figure 9. Perceived relevance of Verificator

Data displayed in Figure 10 imply that 44.30% of study participants perceived Verificator as useful tool for teaching programming. More specifically, 50.41% of students agree that employment of Verificator improves their effectiveness in understanding both syntax and semantics of the C++ programming language (M = 2.78, SD = 1.165). Moreover, 38.02% of respondents believe that they can learn programming concepts more efficiently when they use Verificator than when they employ teaching materials (e.g. textbooks, educational artefacts deployed on LMS, etc.) for the same purpose (M = 2.93, SD = 1.146).

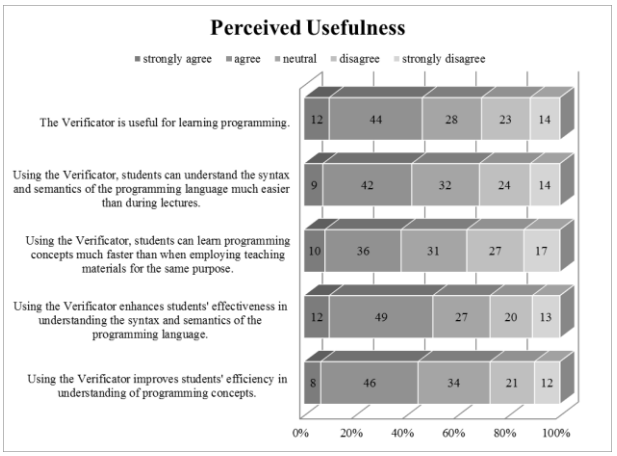

Figure 10. Perceived usefulness of Verificator

As can be observed from the data shown in Figure 11, majority of respondents (66.94%) believe that using the Verificator does not require a lot of effort. Namely, 76.03% of students reported that is easy to learn to operate the Verificator (M = 2.03, SD = 0.921). Similarly, 60.33% of them stated that is easy to become skilful at using the Verificator (M = 2.33, SD = 1.028).

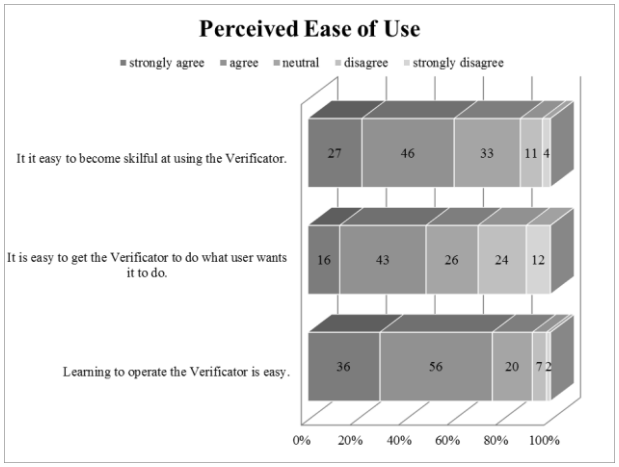

Figure 11. Perceived ease of use of Verificator

As can be observed from the responses to questionnaire items shown in Figure 12, 42.81% of study participants think that Verificator successfully focuses and holds their attention while making the solution of the problem-based assignment. More specifically, 53.72% of students reported that while using the Verificator, they are completely focused on solving the assignment (M = 2.71, SD = 1.294). Furthermore, 30.58% of them stated that nothing can distract them while they are writing the program code in the Verificator (M = 3.16, SD = 1.162).

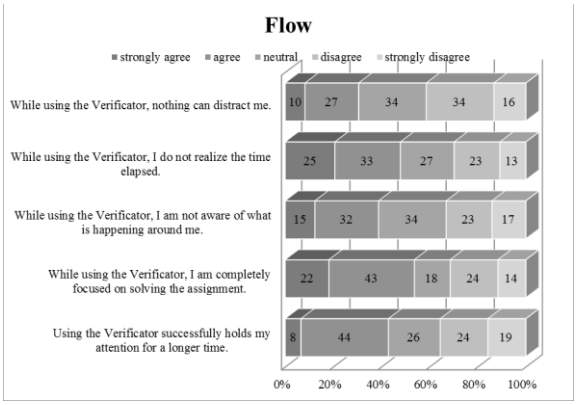

Figure 12. Flow of using the Verificator

Data presented in Figure 13 indicate that majority of respondents perceived themselves as self-efficient in the context of using the Verificator. For instance, only 8.26% of students believe that they could use Verificator only when someone would instruct them what to do (M = 4.07, SD = 0.941). In addition, 47.93% of study participants think that they could use Verificator even if they had not employed similar tools before (M = 2.73, SD = 1.162).

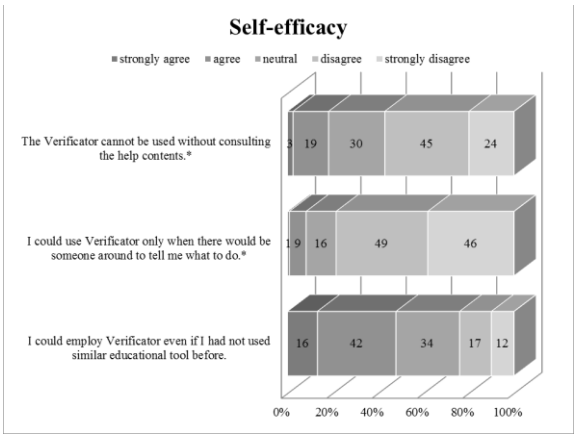

Figure 13. Perceived self-efficacy in using the Verificator

V. CONLUSION

The objective of this paper was to explore diverse facets of the perceived quality in use of the educational tool Verificator. For that purpose a pilot study was carried out during which the post-use questionnaire was developed and administered among information systems students. An analysis of collected data revealed strengths and weaknesses of the Verificator. In that respect, the introduced questionnaire can be employed for measuring the quality of educational tools meant for teaching programming. Moreover, the proposed quality attributes can be used as a background for future research advances in the field. Given that the work presented in this paper is a part of an ongoing research, our future research efforts will be focused on modelling interrelations among determined quality attributes.

REFERENCES

[1] D. Radošević, T. Orehovački and A. Lovrenčić, "Verificator: Educational Tool for Learning Programming", Informatics in Education, vol. 8, no. 2, pp. 261-280, October 2009.

[2] T. Orehovački, D. Radošević and M. Konecki, "Acceptance of Verificator by Information Science Students", Proceedings of the 34th International Conference on Information Technology Interfaces, V. Lužar-Stiffler, I. Jarec and Z. Bekić, Eds. Cavtat: IEEE, 2012, pp. 223-230.

[3] D. Radošević and T. Orehovački, "An Analysis of Novice Compilation Behavior using Verificator", Proceedings of the 33rd International Conference on Information Technology Interfaces, V. Lužar-Stiffler, I. Jarec and Z. Bekić, Eds. Cavtat: IEEE, 2011, pp. 325-330.

[4] ISO/IEC 25000, "Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE", 2005.

[5] ISO/IEC 25010, "Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models", 2011.

[6] ISO/IEC 25012, "Software engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Data quality model", 2008.

[7] V. Venkatesh and H. Bala, "Technology Acceptance Model 3 and a Research Agenda on Interventions", Decision Sciences, vol. 39, no. 2, pp. 273-315, May 2008.

[8] W. H. DeLone and E. R. McLean, "The DeLone and McLean model of information systems success: a ten-year update", Journal of management information systems, vol. 19, no. 4, pp. 9-30, 2003.

[9] V. Venkatesh, J.Y.L. Thong, X. Xu, "Consumer acceptance and use of information technology: extending the unified theory of acceptance and use of technology", MIS Quarterly, vol. 36, no. 1, pp. 157-178, March 2012.

[10] A Bhattacherjee, "Understanding information systems continuance: an expectation-confirmation model", MIS Q. vol. 25, no. 3, pp. 351-370, 2001.

[11] E. C. Sheeson, "Computer anxiety and perception of task complexity in learning programming-related skills", Computers in Human Behavior, Vol. 21, no. 5, pp. 713-728, 2005.

[12] A. A. Anderson, "Predictors of computer anxiety and performance in information systems", Computers in Human Behavior, vol. 12, no. 1, pp. 61-77, 1996.

[13] V. Ramalingam, D. LaBelle and S. Wiedenbeck, "Self-efficacy and mental models in learning to program", SIGCSE Bull, vol. 36, no. 3, pp. 171-175, 2004.

[14] M. Csikszentmihalyi, M., "Beyond Boredom and Anxiety", San Francisco: Jossey-Bass, 1975.

[15] J. J. Beckers, R. M.J.P. Rikers, H. G. Schmidt, "The influence of computer anxiety on experienced computer users while performing complex computer tasks", Computers in Human Behavior, vol. 22, no. 3, pp. 456-466, 2006.

[16] B. C. Wilson and S. Shrock, "Contributing to success in an introductory computer science course: a study of twelve factors." SIGCSE Bull, vol. 33, no. 1, pp. 184-188, 2001.

[17] D. Potosky, "A field study of computer efficacy beliefs as an outcome of training: the role of computer playfulness, computer knowledge, and performance during training", Computers in Human Behavior, vol. 18, no. 3, pp. 241-255, 2002.

[18] H. C. Lane and K. VanLehn, "Teaching the tacit knowledge of programming to novices with natural language tutoring", Computer Science Education, vol. 15, no. 3, pp. 183-201, 2005.

[19] S. H. Chang, C. H. Chou and J. M. Yang, "The Literature Review of Technology Acceptance Model: A Study of the Bibliometric Distributions", PACIS Proceedings, 2010.

[20] J. Webster, L. K. Trevino, L. Ryan, "The dimensionality and correlates of flow in human-computer interactions", Computers in Human Behavior, vol. 9, no. 4, pp. 411-426, 1993.