Teachers' Perspectives on Learning and Programming Environments for Secondary Education

Riko Kelter, Matthias Kramer, Torsten Brinda

Teachers' Perspectives on Learning and Programming Environments for Secondary Education

Riko Kelter
University of Siegen, Germany, Department of Mathematics, riko.kelter@uni-siegen.de

Matthias Kramer, Torsten Brinda
University of Duisburg-Essen, Computing Education Research Group, Germany,
matthias.kramer@uni-due.de, torsten.brinda@uni-due.de

Abstract. Teaching and learning programming is a challenge. Although several learning and programming environments have been proposed for classes, there seems to be more dissent than consensus as to which tools are preferable over others. This paper investigates teachers' perspectives on popular learning and programming environments used in secondary computer science education in Germany. The environments investigated are: BlueJ, Scratch, Greenfoot, Eclipse, MIT App Inventor, Processing IDE, and Alice. Based on prior research, a catalogue of environment features supporting the learning processes of students was constructed. Using these criteria, an online-survey was conducted with computer science teachers in North Rhine-Westphalia, Germany. In the survey, the participating teachers evaluated the selected tools' adequacy for teaching object-oriented programming. The findings support the results of prior research conducted with students, stressing the importance of a simple and user-friendly graphical user interface (GUI) as well as the option to visualise classes and objects. Contrary to prior studies, the results show that teachers do not see the editor as equally important, as students do, and that there is no consensus about the role of the area of application for choosing an integrated development environment (IDE). Student-friendly debugging messages as well as a step-by-step execution of programs were identified as important features. Although no tool excelled for every criterion, the clear favourite was BlueJ.

Keywords. Educational programming environments, teaching and learning programming, object-oriented-programming

1 Introduction

Teaching and learning programming constitute a challenge. In Germany, a focus in secondary computer science education lies on object-oriented programming (OOP) in Java. While there is a variety of suitable learning and programming environments for this task, teachers are free to choose what best fits their personal preference. Given this situation, it is important to find out what exactly constitutes a beginner-friendly environment that supports the learning process. The advantages and problems of common integrated development environments (IDEs) have been discussed by Xinogalos et al. [1], Georgantaki [2] and Uysal [3], all of them focusing on students'

perspectives. While the results show important features from the students' points of view, there is no information available about which features teachers regard as necessary to enhance the learning process. While students' perspectives are often based on a short period of use, most teachers use such environments for a long period of time and therefore have valuable knowledge about common fallacies or benefits of a given environment from a currently uninvestigated perspective. Therefore, teachers' perspectives are investigated in this paper, adding a new perspective to the debate about which aspects are important when selecting an educational OOP environment.

2 Background and related work

Extended research has been carried out investigating educational IDEs. To give a framework for orientation in computer science education research, Hubwieser et al. [4] constructed the so-called *Darmstadt model*, which defines "educational relevant areas". According to this paper, *teaching methods* and *media* are two crucial research areas in computer science education. Hubwieser et al. [5, p. 7] give a definition of what constitutes these areas: *"(12) Media: Technical Infrastructure, (...), Tools, Didactical Software, (...). Which (...) programming languages or environments, personal learning environments were found to support motivation (5), (...)?"* Nevertheless, no consensus has been established on what makes a good educational IDE. Xinogalos et al. [1] stress that there is no universally accepted framework for evaluating educational IDEs. Already in 2002, McIver [6] discussed how educational IDEs should be analysed. McIver [6, p. 2] points out that while there *"is some evidence that a well-designed programming environment can assist students learning to program (...) there have been few, if any, direct evaluations of whether the choice, or design, of programming development environments has a real impact on learning"*. Gross and Powers [7] additionally showed that the results even for environments which have been evaluated, cannot be replicated in a lot of cases. Georgantaki [2] built upon McIver [6] and investigated important features of IDEs. The results showed that the graphical user interface (GUI) and visualisation of program dynamics is important to students. Xinogalos ([8, 9]) showed in multiple studies that the educational IDE used does have an impact on learners' outcomes. To make analyses of environments more standardised, Uysal [3] proposed a framework for evaluating educational IDEs. Xinogalos [10] also isolated multiple important features of programming environments by conducting a study in which students compared a selection of IDEs in terms of effectivity and adequacy for different goals like programming in general, OOP or fundamental object-oriented concepts.

3 Goals

One key aspect missing in prior research is the influence of the learning or programming environment used on students' success compared to other factors. To judge the relevance of results, it is important to know which role the environment used plays at all. Also, while prior studies focused on isolating important features of such

tools based upon students' views, the perspective of teachers has not currently been investigated. Therefore, direct evaluations of learning and programming environments – exploring which framework components are crucial, as well as benefits and disadvantages – from the perspective of teachers are missing. Additionally, there is a clear lack of knowledge about the spread and use of such tools in classrooms. Thus, four research questions have been formulated for the study:

- **RQ1**: What influence does the educational OOP environment have on students' learning success compared to other factors, according to teachers?
- **RQ2**: What features should an educational OOP environment have, according to teachers?
- **RQ3**: Which educational OOP environments are used and preferred for classrooms by teachers?
- **RQ4**: Which benefits and disadvantages exist for selected educational OOP environments, according to teachers?

To investigate the influence of the environments on students' successes in relation to other factors, **RQ1** was formulated. To answer **RQ2**, based upon the results from Xinogalos et al. [1], Uysal [3] and Georgantaki [2], a framework of six components for an educational programming environment was constructed for the study: graphical user interface (*GUI*), visualisation tools (*visual*), editor (*editor*), compiler and error messages (*compiler*), execution system and debugger (*debugger*), and area of application (*area*). Based upon the evaluated tools previously listed [2, 9, 10], six educational environments were then selected to answer **RQ3** and **RQ4**: BlueJ, Greenfoot, Scratch, Eclipse, MIT App Inventor, Processing, and Alice.

4 The study

The chosen format of the study was an online survey, which was answered by 102 secondary education computer science teachers in North Rhine-Westphalia, Germany, from which 79 were filled out completely. The format was chosen to reach as many teachers as possible, as well as making the process of filling out the survey as easy as possible. The 79 questionnaires built the basis of the analysis, out of which 57 were filled in by male and 18 by female teachers (on 4 questionnaires no gender was selected). Fifty-two of the 79 teachers had mathematics as their second teaching subject, 12 had physics, with the rest split equally on other subjects.

4.1 Influence of the environment on learning success compared to other factors according to teachers

The target of **RQ1** was to investigate whether, according to teachers, the environment plays a significant role for students' successes, compared to other factors. Teachers were asked to rate the importance of different factors for the learning success of students. The given factors were the used *API*, the used *textbook*, the used *IDE* as well

as the student's (*student*) and teacher's personality (*teacher*). The used application programming interface (*API*) refers to the programming library used in class to enhance a student's learning process. Just as different textbooks and IDEs exist, there also exist a variety of available *APIs* for teaching OOP (e.g. GLOOP for Java), often with helpful examples and projects for classroom usage. Next to the *IDE*, teachers are free to choose an *API* for their lessons in Germany, so it is reasonable to investigate its influence compared to the selected *IDE* and textbook, as well as the personality of teachers and students. The answers were Likert-scaled, with 1 = unimportant, 2 = rather unimportant, 3 = rather important, and 4 = important. The results showed that while the used textbook seems to play a rather unimportant role according to the teachers, with a mean of 2.26, the used *IDE* and *API* have a definitive impact on students' outcomes (means of 2.84 and 2.90). In total, the student's and teacher's personality are the most important factors for teachers, with means of 3.60 and 3.69.

4.2 Evaluation of framework features

To answer **RQ2**, the identified framework features were evaluated. Teachers could agree or disagree to given statements about educational OOP environments on a Likert-scale including 1 (= don't agree), 2 (= rather don't agree), 3 (= rather agree) and 4 (= agree). The statements were constructed based upon the six selected components, which were based on prior research (see section 3):

A programming environment for learning OOP should

- *have a simple, intuitive user interface,*
- *have the option to visualise objects and classes,*
- *have the option to execute a program step-by-step,*
- *have a code editor, which allows next to the entry of text also puzzle-like code snippets,*
- *provide comprehensive, beginner-friendly error messages instead of normal error messages of the compiler,*
- *have a wide area of application and therefore allow the development of different kind of program type (e.g. animations, web applications, apps).*

The results are shown in Figure 1. The order of components on the x-axis in Figure 1 follows no specific logic. The highest agreement can be found in the categories *GUI* and *visual* with means of 3.73 and 3.61. Similar is the rating for a debugger that allows step-by-step execution of the program (*debug*), with a mean of 3.46. Beginner-friendly error messages have a mean rating of 3.16 and the rating for the area of application (*area*) has a mean of 2.66. Nearly half of the teachers rather agree here, and the other half rather does not agree, so the opinion is split. With a mean of 2.20, the *editor* component has the lowest mean of all six components. Therefore, an editor supporting puzzle-like code snippets is not seen as a necessary feature, according to the teachers' perspectives, contrary to the students' points of view described in Xinogalos et al. [1], Uysal [3], and Georgantaki [2].

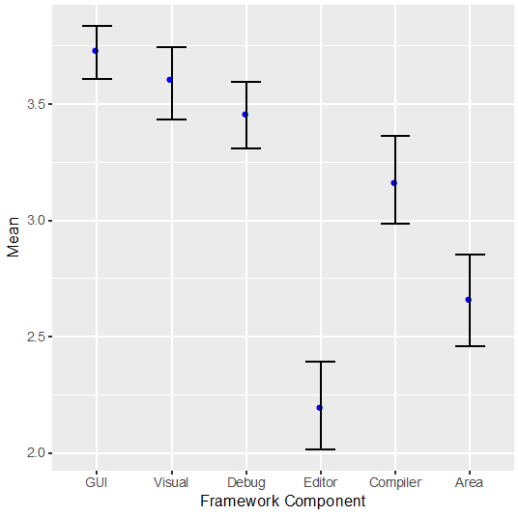

Figure 1. Rating of framework components according to teachers

4.3 Use and preference of educational OOP environments

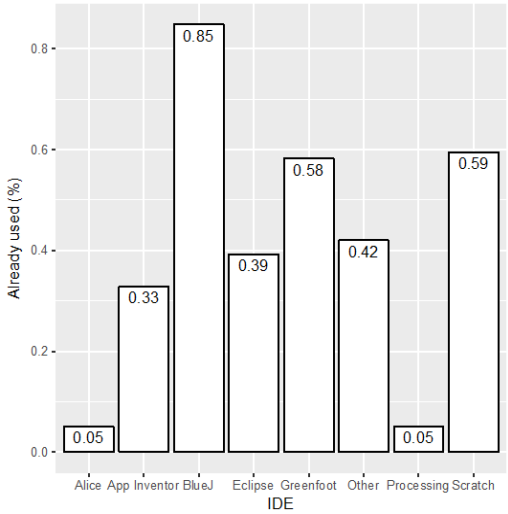

Figure 2. Spread of different programming environments

Figure 2 shows that 85% of teachers already used BlueJ in class. Greenfoot and Scratch have been used by 58% and 59% respectively, and Eclipse has been used by 39%. There were plenty of other IDEs not listed in the survey, and 42% of the teachers did already use some other IDE. This underlines the vast dissent currently existing in the use of

IDEs in classrooms. Teachers listed additionally Java Editor (16 teachers), Netbeans (7 teachers), Java Karol (4 teachers) or the Lego Mindstorms IDE (4 teachers).

When asked for their personal preference for classroom use, teachers' answers showed the distribution in Figure 3. Clearly, BlueJ seems to be the most preferred OOP environment (48), while Eclipse (6) and Greenfoot (8) are preferred by a few teachers.

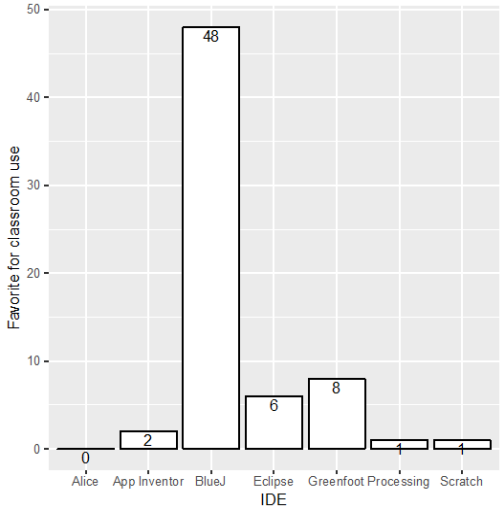

Figure 3. Favourite programming environments of teachers for classroom use

4.4 Evaluation of the selected environments

Depending on the environments they already used in class, teachers had to answer additional questions. If a teacher selected BlueJ, he or she had to rate the following statements on a 5-point Likert-scale using 1=do not agree, 2=rather do not agree, 3=neutral, 4=rather agree, 5=agree (and the same for every other environment, but with replacement of 'BlueJ' for the appropriate name in the following):

- *BlueJ has a structured, student-friendly user interface.*
- *BlueJ offers the option to visualise objects and classes supporting the understanding of the program dynamics.*
- *BlueJ offers the option to execute a program step-by-step to retrace the program flow.*
- *BlueJ offers a code editor which allows the use of puzzle-like codeblocks, to write a program.*
- *BlueJ shows student-friendly error messages easy to understand.*
- *BlueJ can be used for the development of a variety of program types, e. g. apps, web applications or animations.*

Table 1 shows the evaluation of the selected environments with regard to the framework components given in Figure 1.

Table 1. Teachers' evaluation of the selected environments with regard to the framework components

IDE	GUI	Visual	Debugger	Editor	Compiler	Area
Framework-Components	3.73	3.61	3.46	2.20	3.16	2.66
BlueJ	4.12	3.88	3.75	1.64	2.40	2.36
Greenfoot	4.31	4.10	3.64	1.61	2.45	1.92
Scratch	4.37	3.08	3.36	4.78	3.03	2.45
MIT App Inventor	4.06	3.06	2.41	4.72	2.65	3.11
Eclipse	4.84	3.00	4.44	2.04	2.42	4.84

4.5 Evaluation of BlueJ and Greenfoot

The evaluation of BlueJ and Greenfoot can be seen in Table 1. The mean of every framework component is listed for the selected environment as well as the mean for agreement to the framework components themselves, as shown in Figure 1.

BlueJ excels in most categories, especially the GUI, visualisation options and a step-by-step execution option. The environment lacks user-friendly compiler messages and a broad application area, but, especially in the last case, it is unclear if this component is necessary for a good educational OOP environment. BlueJ also offers no editor allowing puzzle-like statements, fitting into the results of Figure 1.

The results for Greenfoot show a similar picture. According to the teachers, Greenfoot has a good GUI, visualisation options and step-by-step execution option. The editor here also allows no puzzle-like code blocks and therefore is rated poorly. However, there seems to be a consensus that such a component is not necessary at all for a good educational OOP environment (see Figure 1). While the area of application is smaller than BlueJ's, the compiler seems to offer a degree of user-friendlier error messages.

4.6 Evaluation of Scratch and the MIT App Inventor

Scratch scores very high on the *editor* component with a mean of 4.78 (see Table 1). Offering its unique puzzle-like code-drag-and-drop system, it excels in this category. The *GUI* is structured and user-friendly according to the results. A weak point according to the teachers is the area of application (*area*). Taking a look at the step-by-step execution option (*debugger*), user-friendly error messages (*compiler*) and visualisation options for objects and classes (*visual*), Scratch is rated as 'good'.

The MIT App Inventor also has a very high rating in the *Editor* category, with a mean of 4.72 (see Table 1). However, it is only used by a few teachers, which may bias the results. The MIT App Inventor's editor seems to be highly capable of supporting students' learning successes by allowing the use of puzzle-like code blocks for building apps. Problematic indeed is the fact that most teachers do not see such an editor as a necessary feature for a good educational OOP environment (see Figure 1 and compare

the mean). The *GUI* is still rated 'good', with a mean of 4.06. While the area of application (*area*) still has a mean of 3.11, there is a slight tendency of teachers not to agree with the fact that the MIT App Inventor allows step-by-step execution to support students' learning processes, as can be seen by the mean of 2.41 of the *debugger* component.

4.7 Evaluation of Eclipse, Processing and Alice

Eclipse differs a lot from the teachers' general rating for the different framework components, as can be seen in Table 1. While its *GUI* and *debugger* are rated high, with means of 4.84 and 4.44, the error-messages (*compiler*) seem not to be beginner-friendly, with a mean of 2.42, and the visualisation options (*visual*) are also rated worse than every other environment with a mean of 3.00. The *editor* also is not capable of using puzzle-like code blocks; therefore its rating is quite low. A definitive strength can be seen with the high rating for the area of the application *area*, which has a mean of 4.84.

For Alice and Processing, there was so little data that summarising them would not have been useful. In total, 3 teachers gave feedback about the Processing environment. Alice was evaluated only by 2 teachers, who basically underlined that it is more appropriate for non-secondary education. Therefore, no data for Processing and Alice are included in Table 1.

4 Discussion and conclusions

The study presented in this paper investigated the perception of secondary-school computer science teachers regarding seven popular educational learning and programming environments, with a focus on OOP, and the features that an ideal introductory environment should have. The study findings were based on an online questionnaire filled in by 79 secondary education computer science teachers in North Rhine-Westphalia, Germany. The results can be summarised as follows. Measuring the influence of the educational programming environment for learners' successes in OOP according to teachers was the target of **RQ1**. The results showed that while the textbook is seen as rather unimportant, the educational IDE used and the API used are seen as rather important. However, the personality of the student and teacher are regarded as even more important. Future research should further investigate how exactly the educational programming environment plays a role for students' successes and how the influence can be separated by the influence of the API used.

The necessary features of educational OOP environments were the target of **RQ2**. The constructed framework of six components was evaluated by the teachers. In summary, teachers did agree with the fact that the *GUI* and visualisation options are highly relevant for a good educational OOP IDE. Additionally, a step-by-step execution system and understandable error messages are regarded as important. Contrary to the results from Xinogalos [1], Uysal [3] and Georgantaki [2], the *editor* component was not seen as crucial and may not be a good candidate for the framework.

The use and preference of educational OOP environments was the target of **RQ3**. The most used OOP environment was BlueJ, but 42% of the teachers had also used other environments. When asked for preference in classrooms, the highest scorer was BlueJ.

The benefits and disadvantages of the environments were investigated in **RQ4**. Clearly, no single IDE excelled in every framework component. However, BlueJ and Greenfoot had 'good' to 'very good' ratings in most components. While Greenfoot seemed to be good as an introductory environment, BlueJ was also seen as a stable long-term solution by teachers, according to comments. Scratch (as well as the MIT App Inventor) were evaluated as 'good', especially with regard to the *editor* component, but they did not score as equally high as BlueJ or Greenfoot in other components. Eclipse was evaluated as the most professional, but also complex environment, and Alice and Processing had too few evaluations to construct meaningful results.

In the next step, the proposed framework of six components needs to be specified further by defining which features exactly make up a 'good' component. While an *editor* component supporting just puzzle-like code blocks is not regarded as necessary by teachers, there may be other features like auto-completion which may be regarded as helpful. Future work should further investigate these structures in detail.

References

1. Xinogalos, S., Satratzemi, M., Malliarakis, C.: Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment? Education and Information Technologies, 22(1), 145–176 (2017)
2. Georgantak, G.I: Using Educational Tools for Teaching Object Oriented Design and Programming. Journal of Information Technology Impact, 7(2), 111-130 (2007)
3. Uysal, M.P.: Interviews With College Students: Evaluating Computer Programming Environments For Introductory Courses. Journal of College Teaching & Learning, 11(2), 59-70 (2014)
4. Hubwieser, P., Armoni, M., Giannakos, M.N., Mittermeir, R.T.: Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools. ACM Transactions on Computing Education, 14(2), 39-51 (2015)
5. Hubwieser, P., Armoni, M., Giannakos, M.N.: How to Implement Rigorous Computer Science Education in K-12 Schools? Some Answers and Many Questions. ACM Transactions on Computing Education, 15(2), 5-17 (2015)
6. McIver, L.: Evaluating languages and environments for novice programmers. presented in Fourteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG), Brunel University, Middlesex, UK (2002)
7. Gross, P., Powers, K.: Evaluating Assessments of Novice Programming Environments. In Proceedings of the First International Workshop on Computing Education Research, October 01-02, Seattle, WA, 99-110 (2005)
8. Xinogalos, S., Satratzemi, M.: A Long-Term Evaluation and Reformation of an Object Oriented Design and Programming Course. In I. Aedo, N.-S. Chen, Kinshuk, D. Sampson, L. Zaitseva (Eds.), Proceedings of the Ninth IEEE International Conference on Advanced Learning Technologies, 64-66, DOI: 10.1109/ICALT.2009.131 (2009)
9. Xinogalos, S.: An Evaluation of Knowledge Transfer from Microworld Programming to Conventional Programming. Journal of Educational Computing Research, 47(3), 251–277

(2012)
10. Xinogalos, S.: Object-Oriented Design and Programming: An Investigation of Novices' Conceptions on Objects and Classes. ACM Transactions on Computing Education, 15(3), 13-34 (2015)