Teaching one Language in More Depth is better than Many Languages Superficially

R. Horváth*

*Faculty of Education at Trnava University/Department of Mathematics and Computer Science, Trnava, Slovakia
e-mail rhorvath@truni.sk

Abstract—**This paper describes the findings on students' problems in programming learning that have been captured through interviews and essays. It briefly introduces the new class which we intend to use in programming teaching.**

I. Introduction

We currently deal with the issue of rebuilding a series of programming courses at the Faculty of Education at Trnava University in Trnava. The programming belongs to the less easy manageable for the students; therefore teachers worldwide and in Slovakia constantly look for innovative ways of programming teaching. We have opened that question, too.

The question is: how to manage programming courses, so that students would have better results absolving them? The greatest chasm within the teachers worldwide is in the selection of first language (and programming environment). Next point is that the programming courses are put into specific time-frame, but master single programming language can take years of study.

We should regard the fact, that significant per cent of students coming up to our faculty to study Computer Science Teaching have never met any programming language before. Worse, many of them have no idea that they will have to learn to program after they join the faculty. If we put too much information to such as student, he will lost the overview quickly, fails in study, and will leave, eventually.

Despite of the chasm in the selection of first language, we decided to use Java as the first language and BlueJ as the programming tool (environment). An interview series with the students showed to us that less acceptable for them is to learn many programming languages and work with many programming tools (or environments) than learn single language and single tool. We decided to unify programming language and tool taught. Java and BlueJ is our choice.

II. Java in Programming Teaching

There are several kinds of beginner programmers – students. Students studying at our faculty are future computer science teachers. They are different than "standard" computer science students. We have to adapt our requirements to their needs. As we mentioned, discussions about proper first language and first programming tool are in progress, for years. The view is changing with the current trends in programming and with new programming languages and environments. Computer science praxis influences the computer science teaching, and so it should be…

As we found out by studying several public information sources ([7], [1], [5]) and performing interviews with several teachers working at universities in Slovakia, nowadays, three groups sustaining different programming languages in computer science teaching are formed. The supporters of each group have their own opinions, and it is very hard to resolve the dispute. They are as follows: 1. to teach using Pascal or Pascal-like programming language (Pascal, Object Pascal, Delphi…); 2. to teach using C-like programming language (C++, C#, Java…); 3. to teach using different language – this group is autonomous and prefers completely different languages as Python [9], Lisp, Scheme etc.

Pascal is, for example, a language designed for teaching. It is a language of verbal keywords and operators (e.g. begin, end, procedure, and, or, mod…). However, all these keywords are in English. Experiences taken over years showed us, that English is, despite the wide-spread in computer praxis and computer science, not routine for the Slovak students. That means that human readable keywords of operators and programming structures don't play such important role for Slovak students as some other factors. Languages using C-like syntax are more symbolic (&&, ||, ++, +=, etc.). These symbols have for most Slovak students the same meaning as English human readable keywords. Students just have to memorize them… Currently, we have no solution for this issue. We can just say that this is not the key factor for the choice of language taught. Personally, I can say that foreign language makes no problem for a man, who wants to learn to program. I, for example, started to learn programming in Basic (programming language using pure English keywords) with help of German written manuals as eight, maybe nine-year old boy, with having had no idea about syntax, algorithms, and with zero knowledge of English or German. But… often we are forced to teach people, who don't want to learn to program. (That's the point…)

Let us take a small thought about transition between programming languages during study. Surely, the transition process is serious problem just like the syntax of programming language is. Our past students had no other choice, just accept the fact they must change the programming language almost each two terms. We have noticed that this is a problem for them, and we convinced of this through interview series with the students. Several students said: *"We have had to teach several times the programming from the ground… You suppose that we are able to program in all these languages, but the fact is that we are hardly able to write any basic algorithm in some of these languages…"* We could not ignore these expressions. So we decided to teach one language in more depth than several languages superficially. One question stayed open for some time period: which language? We knew that transition from C-like language to Pascal-like language is easier than the reverse. That has indicated to teach some C-like language to make the students prepared to the reverse (easier) transition way in the future, after leaving our faculty… And after several another recommendations the faculty took the decision to teach Java. We can say that the final choice of the starting programming language taught at our faculty has been taken with regard of other benefits of the language chosen.

Joel Spolsky published an article full of criticism against Java. He says, what happened to the programmers' hard work? [6] He says: *"Java is not hard enough to be used to discriminate between great programmers and mediocre programmers."* And more: *"...the ability to understand pointers and recursion is directly correlated with the ability to be a great programmer."* Sure, this is a good point for computer science students, but is for future teachers too? Spolsky is talking about permanent decrease of requirements made for the students. Yes, there are no doubts that with teaching someone who, as we expect, will develop commercial software, we take the responsibility to teach him how to make good programs, good algorithms, and how to be a good programmer. But with future teachers we are in different position. Our students have problems with more basic language features, they don't miss pointers, and I think they would be thankful to Java for covering pointers deep under layer visible to programmers, if they knew what the pointer is. I think Joel Spolsky is right, but with "classical" computer science students. Maybe there should not be a differentiation between "classical" computer science students and future teachers, but it is, nowadays. And I think it will be for a long time in the Slovakia (and some near countries, too; if anybody wants to know more about our school system, please, write to me…). We should, first of all, produce teachers that are able to program, and, at least, able to create basic algorithms.

I am not strong Java supporter. I did not know Java before 2009. Until then I was able to compare only Pascal and C/C++. After meeting Java, I was able to compare it with other two, and also able to better understand some claims that Java is more close to (Object) Pascal than C/C++. Java has C-like syntax, but I see something different in it. I think it is better designed. The whole language philosophy is different than the C/C++. It is strictly object oriented, and has strong object basis with optimized algorithms, which can be used by beginner and professional programmers equally. Java claims to be simple, object oriented, familiar and more [8]. We decided to teach Java, but more important is that we decided to teach single language in introductory and following courses. So the students have better chance (more time) to learn about how to make programs before they meet some other language (and before they are forced to invest the energy into the transition). Maybe/I think the decision with Java was not as bad. For the students will be less painful to go off from Java to simpler language than the reverse. But we have now the responsibility to make introductory programming courses motivating enough.

III. Anonymous Essays and Interviews with our Students

If we want to find answers for some kinds of questions, we have to ask. In the anonymous essays students should to write their opinion on programming teaching at our faculty, the strengths and weaknesses they are able to see. Also, the teacher has made many interviews with the students, whose learn programming at our faculty now, or learned it in the past. He tried to find out, what are the most problems in the programming learning. It has been made in academic year 2009/2010.

Mostly, the students wrote about "big problems in understanding" the teacher's explanation. They noticed that "only few schoolmates really understand the matter." Apparently, they often talked each other about the problems with programming learning. The biggest problem was to "catch the train." I think that this can always be a problem for few students, but not for the most of the classroom. After this happens to a student, it is almost impossible to get in and learn continuously. Different students wrote about different parts of the curriculum which they don't understand, finally each part of the curriculum has been mentioned at least once. After more detailed analysis of the essays we tried to get the most problematic parts. Ironically, they were exactly the elements we worked most often with. The tireless repeating of the language basics (branching statements, loops, creating the conditions, the methods definition…) seemed to have no effect on our students' understanding. Only simplest language elements (as "if" and "if-else" statements) were mentioned as really easy understandable.

Doubtless, the students need time to understand even basic language concepts. Indeed, the subject number two in the essays was "the time." Students often asked to "slow down the tempo." They often complained about the fast progress. They proclaimed that they need time for thinking about discussed problems. Although they did not forget on some situations, when teacher asked all students to think about some problem, but it was not often enough for them. Accepting such kinds of requests the tempo has slowed down, but students again proclaimed that the tempo is still too fast. On the one side, we understand this. There is an effect called Cognitive Overload ([4]; it is related to Cognitive Load Theory, which Mark Guzdial mentioned in an article about introductory programming courses [3]). We should avoid letting our learners suffer from it. Have enough time for thinking is important, we agree, but on the other side there is no straight solution. Programming courses have specific time-frame. We are unable to teach all the necessary subject matter with slower tempo and it is not possible to get more time for programming courses. We could give more stress on homework, and we could (again) try another way of teaching. For example, try to apply some good elements of the Flow Theory [2] by Mihály Csíkszentmihályi. It is always good to have clear goals and immediate feedback… We would like to experience if and how would be this applicable in our teaching, but the final success is mainly on the students themselves and their inner attitude.

Another often mentioned problem was "the students' inequality." Most of the students asked to slow down, but several of them asked the right opposite. They claimed "we are moving like a snails," and complained about slow schoolmates doing too much typos. They showed not much understanding to the slower classroom mates. This is unavoidable – experienced students will be bored and beginners will claim that the tempo is too fast… Some students were able to fully reflect that they need more autonomous action – at home alone or with the help of the friends or family members (e.g. more experienced brother). Another few complained about hidden "illegal activities" during classroom training, like Facebook browsing and other amusements… This is inacceptable. We will draw the consequences in the future.

Individual problem is the memorising whole programs or their parts without understanding them. Some students admitted this, and we have observed such manners too. That problem is connected with the distorted concepts of computer science. Many students were surprised by the fact that they should to learn to program. They often have no idea what it exactly means "to program." Maybe at least this fact will change in the near future…

The last information taken from the essays was the "lack of Slovak study materials." Changing this needs just time and hard work to make such materials. We constantly work on new materials, because this is one of the most objective students' requests.

The interviews confirmed some information gained from the essays. Again, the key points were:

- the lack of time – students considered the two terms introductory programming courses as insufficient;
- the chasm between what the teacher wants to explain and what really students understand – experienced students said that the explanation was simple enough, and beginners said that it was too professional, technical;
- the inner motivation – many students do not step into this study programme with the correct concepts, their concepts were often distorted; after realizing that they will have to learn to program they become being demotivated;
- the team – it is important to support the team building – experienced students should help to the less experienced, students should cooperate – helping each other to learn.

These and some other information were gained from the students. Some experienced students told us, they learned by modifying existing programs. They suggested that incomplete examples or programs ready to modification could be given to the beginners as an exercise. This is not a bad idea. We have heard about such techniques [3], but never tried it until now.

IV. The New Plan – Use New Graphics Robot

All the circumstances consecutively led to creation a new tool – Graphics Robot. The Robot is a Java class and it is a successor of the "Korytnačka" class (which means "turtle" in Slovak language; development of Korytnačka started approximately in august 2010). Robot class does not have anything to do with original java.awt.Robot class. It is independent class made for graphics oriented programming partially similar to Logo. The name has been changed from Korytnačka after dishonour original organic name by students, and after advancing the class functionality far behind the borders of original (simpler logo-like) philosophy. The new name sounds international, but this is a coincidence. In fact, the whole class is written (like its predecessor) with regard to Slovak students. All methods and properties are named in Slovak, and whole documentation is written in Slovak.

The simplest Robot's feature is that it can paint a line during movement, just like Logo turtle, but it has many other features against the original turtle. The Robot is able to work in both Cartesian and Polar coordinate systems. It accepts absolute and relative coordinates, paints images, plays sounds, contains tools for basic events handling (mouse, keyboard, timer…), can do some primitive automated actions (like uniform straightforward movement, accelerated movement, and uniform or accelerated rotation), can generate some basic Java shapes (dots, circles, polygons, rotated ellipses, squares, and rectangles…), draw the outlines of generated shapes or fill them, paint texts, work with text files, and much more. Many methods are polymorphic, some are getters, other are setters, and some Robot methods are intended to be overloaded – they are invoked automatically in specific situations and the Robot changes its behaviour after overloading them. We can use all that to better explain the object oriented features, like the method overloading, polymorphism, and their purpose. The new Robot is intended to be used in object oriented programming teaching.

Eventually, here is an example showing how to program a robot to follow last mouse click:

```
rýchlosť(10, false);

new ObsluhaUdalostí()
{
    @Override public void klik()
    {
        cieľNaMyš();
    }
};
```

Ironically now we have to translate some Slovak words back to English to make the example understandable, because the Robot (as we mentioned) is written using Slovak language. The same example in English would look like this:

```
velocity(10, false);

new EventHandler()
{
    @Override public void click()
    {
        targetToMouse();
    }
};
```

The example must be enclosed in class definition that extends (inherits from) the Robot class. The fastest way is to put it in the constructor.

I. Conclusion

The programming teaching is wide area. There are many ways how to organise introductory courses. We described our situation, the decision made at our faculty, and the new tool (Java class package) developed to cover that decision consequences. Programming probably never comes to be smooth for all different kinds of students, but we can always try to make this process as motivating as possible. The students which are interested in programming will appreciate it, and for the other students (not enough motivated) may be this at least a welcome step.

Acknowledgment

The publication of this paper was supported by grant VEGA 1/0247/10.

References

[1] Antonič, M.: *Delphi namiesto Javy.* [online]. http://miso.blog.matfyz.sk/p12296-delphi-namiesto-javy. Accessed: 29. Feb 2010.

[2] Csikszentmihalyi, M.: *Beyond Boredom and Anxiety: Experiencing Flow in Work and Play.* San Francisco : Jossey-Bass, 1975. ISBN 0-87589-261-2.

[3] Guzdial, M.: How we teach Introductory Computer Science is wrong. In *Blog at Communications of the ACM. Trusted insights for Computing's Leading Professionals.* 2009. [online]. http://cacm.acm.org/blogs/blog-cacm/45725. Accessed: 28. Feb 2010.

[4] Herrod, J.: *Cognitive Overload.* 2000. [online]. http://www.jchconsulting.com/fall2000/. Accessed: 1. Aug 2011.

[5] Kotrbčík, M.: *Hocičo namiesto Delhpi a Javy.* [online]. http://mito.blog.matfyz.sk/p12304-hocico-namiesto-delhpi-a-javy. Accessed: 29. Feb 2010.

[6] Spolsky, J.: *The Perils of JavaSchools.* 2005. [online]. http://joelonsoftware.com/articles/ThePerilsofJavaSchools.html. Accessed: 28. Feb 2010.

[7] Ščerbák, G.: *Java namiesto Delphi.* [online]. http://gscerbak2.blog.matfyz.sk/p12293-java-namiesto-delphi. Accessed: 29. Feb 2010.

[8] *The Java Language Environment.* [online]. http://java.sun.com/docs/white/langenv/Intro.doc2.html. Accessed: 6. Apr 2010.

[9] Zelle, J. M.: *Python as a First Language.* [online]. http://mcsp.wartburg.edu/zelle/python/python-first.html. Accessed: 28. Feb 2010.