Students perspective on improving programming courses

Michal Blaho, Martin Foltin, Peter Fodrek, Ján Murgaš

***Abstract*—**Improvement and modernization of the universities courses should be every year's task. This task is especially needed for computer programming courses where new technologies are coming very often. In the first classes of study at the university many students have hard time in programming courses. It is because they comes from different secondary schools and they have different experience of computer programming which is often not enough for university courses or they hadn't programming at secondary schools at all. We did small questionnaire at the Faculty of Electrical Engineering and Information Technology of Slovak University of Technology in Bratislava to know how student evaluate our programming courses and what they want to improve.

***Keywords*—**Diversity, learning, programming, computer skills, students, suggestions.

I. Introduction

DEMAND for high quality engineers is at high level today. For example Germany needs more than 30000 engineers [1,2,3]. Modern engineers must have good theoretical knowledge and practical experience. Employability skills are needed to get, keep and do well on job [4]. One of many skills that are necessary in praxis for our students is know how to do computer programming [5] or software engineering.

Many teachers agree, that many students have problems dealing with the learning of computer programming [6,7]. Teaching style of computer programming is usually individual for each student, therefor is almost impossible to choose right style for computer programming course. Great help with this problem are many information sources (like books, Internet or technological clubs usually named HackerSpaces) that students can use for computer programming during learning process [8].

Manuscript received August 10, 2011: This work was supported by VEGA agency under contract number 1/0592/10, and KEGA agency under contract number 032STU-4/2011.

Michal Blaho is with the FEEIT, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic (phone: 00421-2-60291405; e-mail: michal.blaho@stuba.sk).

Martin Foltin is with the FEEIT, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic (e-mail: martin.foltin@stuba.sk).

Peter Fodrek is with the FEEIT, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic (e-mail: peter.fodrek@stuba.sk).

Ján Murgaš is with the FEEIT, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic (e-mail: jan.murgas@stuba.sk).

What we can do to improve out teaching of computer programming is to incorporate modern learning strategies or methods into listed process [9]. Collaborative learning [10,11] can be very helpful for this process because group of students is forced to work together to achieve same goal. This type of study is similar to teamwork [12,13]. For students it is also beneficial to gain other professional skill like leadership, communication and global awareness. Students search all contents online therefor e-learning is perfect way to obtain their interest in learning. Study materials are also accessible from any location so they can learn almost everything everywhere [14,15]. Students are also to achieve skills from using team coordination software packages known as time tracking systems. They also achieve skills from using source code management systems.

Even most modern learning methods would not be helpful if we are not able to say that methods are effective and suitable for majority of the students. It is very important to ask students what they think about our courses every year [16,17]. Sometimes students have very good ideas to improve our courses via their answers in questionnaires.

This paper is divided into several sections. In the second section we are to write about questionnaire at our faculty. In the third section we point to diversity of student that come in to study at university and their experiences. In the following section we are to write about computer programming teaching at our faculty. In the fifth section we give our suggestions to improve computer programming teaching process for our and other universities. In next section we are to write about interesting courses and last section contains suggestions from students to improve computer programming teaching process at our faculty.

II. Questionnaire

In automatic control there are two main ways how to control systems. Open loop controller doesn't observe how system reacts to input. This control system cannot correct any error that it could make. On the other side the feedback control measures system output to enable to correct errors. It is clear that we mention to apply this knowledge also in education. In fact we do it without planning to do so. If we focus only on modern technologies and adapt lectures to use them we would have current topics but we will not know how students react to applying them. That is why there is important to ask students how they accept our courses methods and content. We could correct any errors we made because we achieve feedback. This task is very important and we should do this at end of term for every year.

We wanted know how students evaluate their computer programming courses at out faculty generally. Small questionnaire was prepared for them with several questions about their background, computer skills and their suggestions dealing with computer programming. Many students at the faculty had interest to improve quality of computer programming teaching process. In a first day of our questionnaire through Facebook social network and university information system we had over 500 submissions (almost 20% of our students). We cannot underestimate student's opinions and their interest for improving courses and the final effect of their satisfaction for us teachers.

The questionnaire was free for any students but we wanted to know class of their study. On the figure 1 you can see distribution of students based on the class of their study. Most students are from first class of study (almost 30%) and third class of study (about 20%). Smallest amount of students were from last class of study but this student was experienced and gave us really good suggestions for this reason.

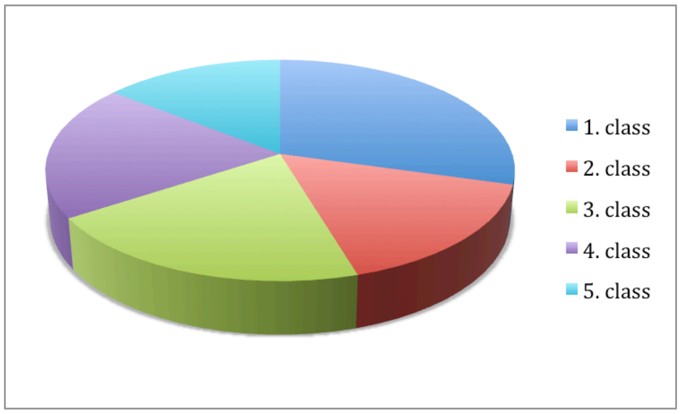

Fig.1 Student's class of study distribution

III. STUDENTS DIVERSITY

Students came from different parts of the country to study at the universities. They have been studying at various secondary schools with various focuses on topics. We have more than 250 secondary grammar schools which are focused on general knowledge with about 90 000 students in Slovak republic. We also have more than 500 secondary technological schools with focus on engineering with about 180 000 students in Slovak republic.

The one of the most common problems in early terms on faculties is this student's diversity. Because they studied at the different schools they have different basis in mathematics, physics, computer programming, technology, etc. The teacher's job is to reduce gap between students in the part of the knowledge necessary for studying at the faculty.

Before we start asking questions about computer programming teaching at the faculty we wanted to know student's background in computer skills and computer programming achieved at the secondary schools. We focus mainly on operating system usage and computer programming languages.

A. Operating systems

Almost every student basically from primary school starts dealing with a computer. In our praxis (industrial informatics) variety of operating systems is used, for example Unix based systems in embedded systems or real-time control. It was in our interest to find out what operating systems students had deal with before study at faculty.

The operating system with vast majority of the market share is Microsoft Windows. It is also well known between students and 99.7% answered that they are familiar with this operating system. Open source Unix based operating systems are growing in popularity because they are free and have near same functionality as Microsoft Windows. Students know these systems and 46% are familiar with Unix based operating systems like GNU/Linux distributions as Ubuntu, OpenSuse or Debian. Apple operating systems are known for support for students and study process. In our country they aren't much spread because the expenses but 10% of the students have experience with this operating system. Other operating systems are used by 3% of the students.

B. Programming experience

Students come to university from different secondary schools and have different knowledge background as we mentioned before. We have found this fact in our courses (mainly in computer programing) when some students can understand lectures and practices easily and some have large level of problems. The difference between students is often enormous. We were curious how students are prepared from secondary schools and if they had been to learn computer programming after all. If they have been learning computer programming by them is another question.

1) Computer programming at school

Teaching informatics in secondary schools is certainty. But the question is, if the secondary schools learn how to design computer programs. On the question, if students had course of computer programming 90% answered positive. The rest 10% haven't got any programming courses yet.

This doesn't mean, that 90% of the students understand principles of computer programming well and are good at algorithm design understanding. Many of them had various teachers with various learning methods or programming topics. Some secondary schools prefer different programming languages then others. We try to find out in our questionnaire what programming languages they have been learned in secondary school.

The most popular language for teaching programming at secondary schools is Pascal. About 71% students have learned this language. Delphi is similar to the Pascal, which is introductory to objective programming and has better graphical user interface capabilities. Delphi is familiar to 10% of the students. Modern programming courses (mainly at universities) starts programming courses with C or C++. About 25% of students learned those languages at secondary schools so they have good chance to pass exams. Technological secondary schools teach low-level programming language Assembly. More than 30% of students learned this language that is used for programming microcontrollers in industrial informatics. More and more secondary schools start to learn more and more popular web technologies like Html, Php or Javascript. Students like this technologies because they are relatively easy to learn. About 21% have been learning web technologies at secondary school. The most popular and used objective language Java has learned only 1% of students. This is depicted in figure 2.

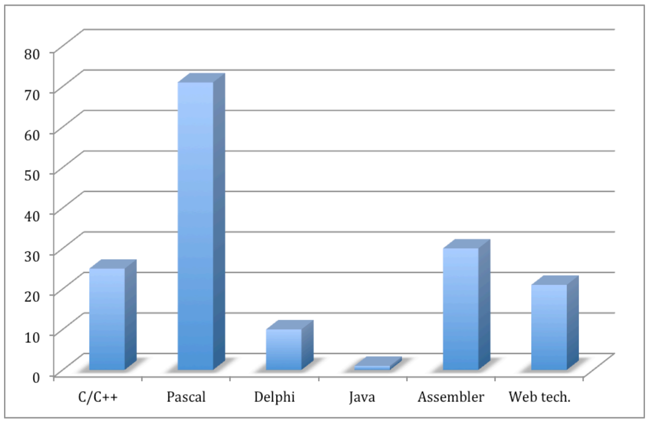

Fig. 2 Programming at secondary schools

2) *Programming in free time*

You don't necessary learn what you want to learn at secondary school. Many students are curious and therefor learn some computer programming languages by them self. These students often achieve better results and have deeper understanding of computer programming languages rather than students to learn only at school. They wanted to learn, but they don't need it, which make the difference.

As a contrast to the school programming free time programming has different distribution of students that learned programming language. Near half of the students (42%) learned web technologies in their free time. It is because the web technologies are simple to learn and are interpreted and student can see results of their work right away. The next favorite programing languages for the students are C/C++, near 27%. Pascal is also popular but not as much (11%). Java is the last known popular language with 10%. Students know that Java is very often used in praxis so they want to learn this programming language, but objective oriented programming isn't as easy as web technologies for example. Other mentioned languages in previous part achieved less than 10%. On the next figure you can see distribution of programming languages that student learned in their free time.

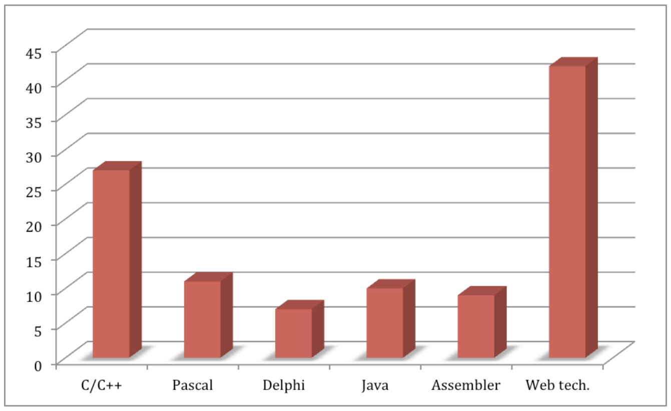

Fig. 3 Programming in free time

3) *Comparison*

If we compare what students learned at the secondary school and what they learned in their own free time, we will see what languages and programming paradigm they know best. Secondary schools learn mostly procedural programming (Pascal, C/C++). Technical secondary schools also learn assembly language because it is needed for technical praxis. More and more secondary schools start to learn web technologies. Students focus mostly on web technologies because they can learn them easily. They also like to learn procedural (C/C++) and objective oriented programming (Java).

IV. Programming at Faculty

As we mentioned before, students with different knowledge of programming are coming to the faculty. The first few terms can be hard for students that hadn't programming courses at the secondary school.

A. *Problems in courses*

We wanted to know how students see difficulty of the programming courses at the university. We ask them, how big problems they had in computer programming courses.

Answers were divided into five groups by problems degree. Major problem had almost 10% of the students. They wasn't capable understand most of the lectures or practices. Above average problems had 21% students. Average problems had almost 35%. These problems are usual on every course. Problems beyond average had 20%. About 13,5% of the students hadn't any problems during computer programming courses. These students came to faculty well prepared for computer programming courses.

The distribution of answers is standard Gaussian like distribution as you can see on figure 4. For this reason we asked this question universally and it applies to all programming classes at the faculty.

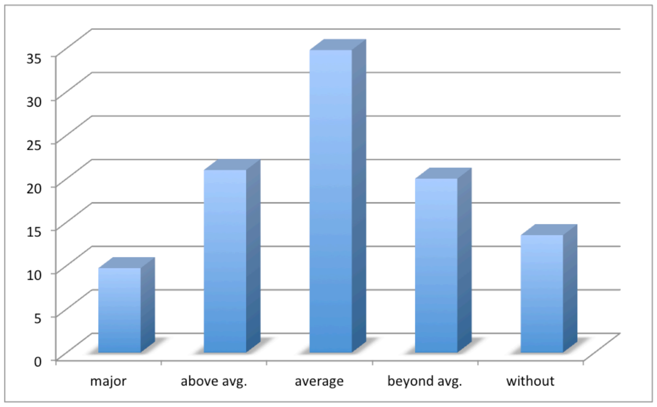

Fig. 4 Degree of problems

Next we ask student which courses of the computer programming language caused them major problems. We chose five programming languages that most students learned. Students had biggest problems with C/C++ programming language courses (more than 40%). This language was to teach in first few terms of study and it is hard for students that hadn't any programming course at secondary school to past them.

Next big group of students selected Matlab courses (about 35%). Matlab is not hard language and it is very similar to C/C++ and also provides capability to objective program constructions. The problem is that many Matlab courses associate theory (for example statistic, numerical methods, control systems) with construction of scripts or schemas and therefor it is hard for students to learn them and apply them using Matlab.

Smaller group of students find hard to learn Java programming language (just about 13%). Many students that find structural programming easy have hard time to switch to object oriented programming and doesn't understand when is beneficiary to use its constructions. They prefer write programs as set of functions. This is because they were not to teach about different paradigms of the computer programming before our courses. They think that there is only way of the computer programming, but this is false assumption.

About the same amount of students like in Java case find hard to program in assembly language. Low-level programing is different than the classical programing and student who can program in C/C++ or Java can have trouble here. The main problem here is lower level of abstraction then in another languages. Assembly language is directly connected concrete hardware.

Smallest group of students (about 3%) had problems with web technologies. On the figure 5 you can see percentages of the problems for computer programming courses.

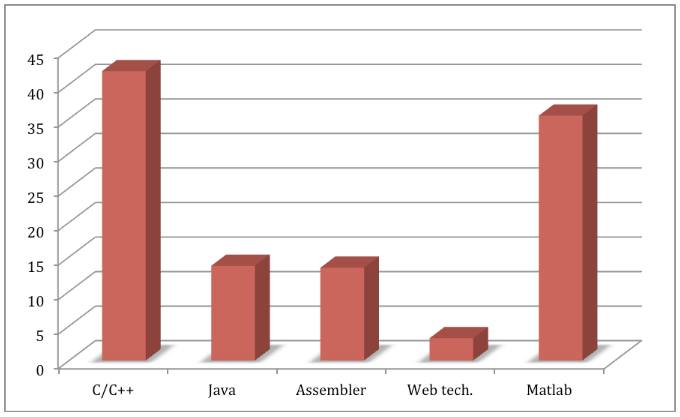

Fig. 5 Problematic courses

In the next question we ask students what computer programming courses didn't make student trouble at all. Again the possibilities were C/C++, Java, Assembly language, web technologies and Matlab.

As in previous question most students selected C/C++ programming language (about 45%). This half of students on the other side had experience with procedural programming mostly in C/C++ or Pascal. Therefor this course was easy for them even it was in first few terms of study. This was not predicted because C/C++ programming languages are often mentioned as very hard programming languages to learn. As we can find out our students have experience with similar languages. This can help students achieve better results in these curses at all.

Also Matlab course had many students thouth that this course was relatively easy to them (more than 36%) to learn. These students could very easely connect theoretical study with practical computer programming and simulations. Some students from technical secondary schools had been to learn Matlab in special courses.

The third course with no problem for student was assembly language (more than 28%). Most of these students was also from technical secondary schools where students produce programs using this programming language so they didn't have problem with this course.

Web technologies courses didn't make any problems for almost 27%. We mentioned one reason for that and it is because they are relatively easy to learn. Another reason for this you can see on figure 2 and 3 where most learned programming languages for secondary school students in free time were these technologies.

The last most popular programming course was Java course (about 20%). As we mentioned for student is hard to switch to object oriented programming, but semantics is similar to C/C++ and therefor it is easy to learn for others.

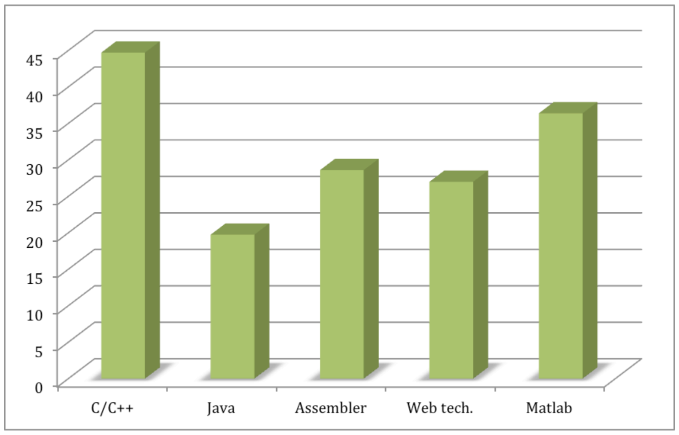

Fig. 6 Non problematic courses

B. *Flipped Gaussian like distribution*

Dehnadi and Bornat [7] point out that student can be divided into two general groups of students. For one group of the students it is extremely difficult to learn computer programming. For other group of the students it is much easier to learn computer programming at all. This group was successful in programming and found it easy. This can be plot on distribution figure of exam grades. One peak is in right half where are placed students with no problem with computer programming. Other peak is on left where are placed students with serious problems in computer programming learning and they fail at exam test. Between this two groups there is placed minority of the average students.

Their mental models cause this. For computer programming there is need for so called consistent group. Consistent group means that all the members of this group use same model of mental translation of the problem. Same model means that they used same type of the solving process for every problem they deal with. It is only assumption of the success in the computer programming courses to be member of the consistent group even if the solving process is not suitable for solved problem.

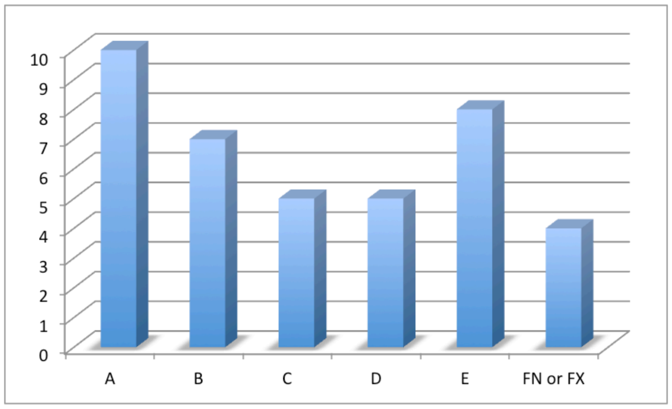

Fig. 7 Unix programming results

Exam results from some courses confirm Dehnadi and Bornat theory. We looked at results from last year's courses about UNIX based system programming and Java programming. In Unix programming course (figure 7, FN or FX means absence at exams or failure) student results have two peaks. One major group is over A, B grades and another above E and fail students. The smallest group of student is above C and D grades. This course was evaluated as one of the most difficult courses to learn by students. Therefor we were not predicted such high rate of the students achieving left peak results.

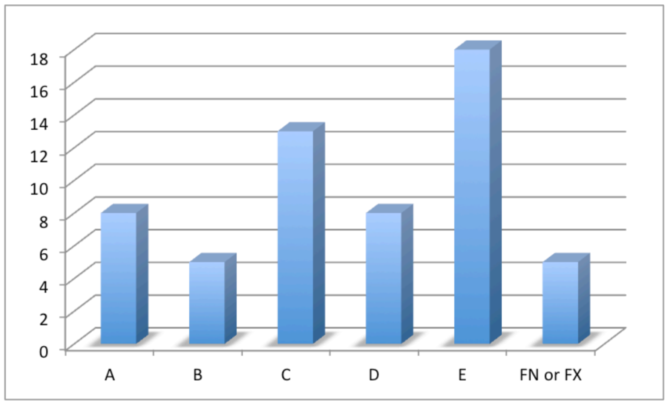

Fig. 8 Java programming results

Java programing course (figure 8) have three major peaks of results. Figures show, that students had in fact problems with passing course or fail at exams. We can see this results in other way. If we forget about grades B and D grades we will see that figure has only one peak and is about E grade. Therefor this course was very hard for most of all students.

V. Our suggestions

It is hard to suggest universal methodology for teaching computer programming. In our case it look like following suggestions may help to improve teaching and learning processes.

A. Experienced students

For experienced students we consider students that could pass final exam at the beginning of the course. In first terms of study number of these student isn't big but grow as years of study increase. Talented students with practical experience are usually these students. Many of them already work on small project part time. We could give individual project to experienced students so they wouldn't have to spend time on lectures and practices on basics topics that they already are able to use.

In later years of study students work on small project outside the faculty as we mentioned before. We could use these students for preparing modern lectures and practices on topics that they work with in their praxis. For example in web technologies some students have sometime better knowledge of open source projects or frameworks than lecturer. Lecturer with help of these students could incorporate this knowledge into few lectures and it would be also interesting for other students. Experienced student could lecture about this topics too and gain their own presentations skills. This helps them in the future carrier. Self-presentation is main disadvantages of the most skilled computer programmers in our country. This is because students have no spoken exams but all exams in their study are in the written form.

Last suggestion is to use experienced students as personal consultants for less prepared students. The main advantage of this is that they will probably explain some different topics from another perspective than the lecturer. This approach can be also risky and we must supervise them because they could do some practices for less prepared student. This principal is processed at our faculty yet. It is caused by the fact that our students are no enough competitive and there is no advantage from being best students of the class. This is cause by no university fee at standard duration of the study at the public universities at our country.

B. Weaker students

Suggestions for experienced students could remove the left peak from figures about programming results. The peak is removed only for lectures and practices but will be shown again at grade results (we must assign grades to experienced student as well). But now we can redesign course to better-fit less prepared students requirements. Some might say that we are lowering standards but the amount of topics should not change. What will change is only form of explanation.

We could use modern methodologies of education like collaborative learning suggested and they are practically proved by many teachers. With collaborative learning student can explain some topics to each other and see topics from another perspective than teacher's perspective. We could use modern communication media like Facebook social networking for this purpose. Collaborative learning is also bind with teamwork where small group of student work to achieve same goal. Work in team can cause small competition where weaker student want to improve so he/she would not be worst in the group. It also can be used to teach team collaboration technologies and software repositories. These technologies was started to explain to some of our students last year and will be improved in the next year.

For weaker students is important to have rich amount of study materials. It should be our responsibility to prepare them. Today almost every student search study materials with search engines on the Internet. Therefor online content should be certainty for us. Classification may to be done offline not online at the practices as it is done now. This means that students upload their programs to server and lectors may study programs at home.

Continuous evaluation is also beneficial for learning purpose. If we evaluate them only on one or two test we wouldn't see what cause them problems. If we evaluate and point their work every week then we will see what is major cause of problems for them. With knowing all of their problems we could react to them on the next lecture in first few minutes for example.

VI. Interesting topics

After questions about secondary school programing and programing at the faculty we asked students what other topics we could focus on. We let them chose from:

- Algorithm basics
- Law for informatics
- Team work
- Work with multiple operating systems

On the next figure you can see percentage of students that would like to study these topics.

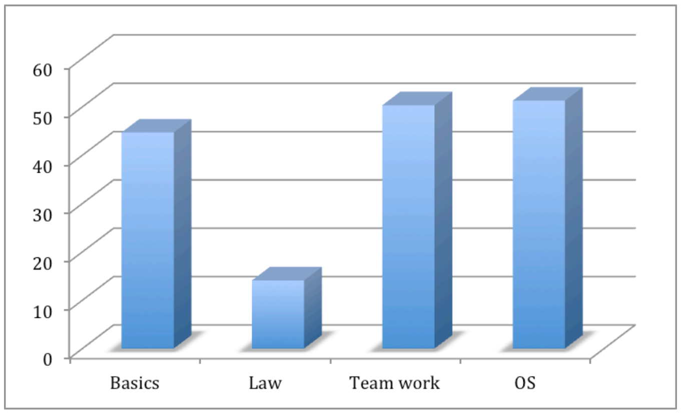

Fig. 9 Students are interested about

About 45% of all students demands better algorithm basics. These are mainly the students that didn't have programming courses at secondary schools. For them is hard to learn new programming language and write basics algorithms. Many courses learn only semantics but small amount of topic are covered from algorithm design perspective. This bring us to the idea that we need teach algorithm design in first few terms probably independent on programming language.

Algorithm design depends on higher-level mathematics that students never had been learned before university classes. Therefor adding algorithm design to the curriculum will force us to add much more mathematics courses to it. But mathematics is highly unpopular for secondary school graduates. Adding mathematics to the curriculum lowers level of the students that will be interested to study at our faculty. Need of mathematics for computer programming is mainly described in D. E. Knuth's book named Art of programming.

We often work with informatics law and we even don't know enough about it. It is mainly pressing "Accept" button for us by installing new software. It would seem that 15% of students is small number but this topic is interesting for future software engineers. They want to know what various licenses mean (for example the GNU General Public License - GNU GPL) and what can do with software legally. Protecting created application from them is also necessary to know. We are preparing translate of the textbook Legal aspects of the information technology to the Slovak language. This textbook is introduction to the author's law and copyright law for information technology experts.

In large software projects there are programmers divided into teams, which works on the different task together. At programming courses many times students work alone. That is good for learning basics but is not enough for praxis. Students demanded courses about versioning and a revision control systems for example subversion, which are widely used in team programming. Another topics are computer programming management and job scheduling. This improves time scheduling for project tasks and together with Gantt graphs are very helpful in large projects. These topics demanded about 50% of all students. As we mentioned we started to teach very basics of these technologies last year but not at desired level. We plan to raise level of these technologies to teach during coming years.

As you seen in question about operating systems the most known operating system is Microsoft Windows. This operating system is unfortunately dominant in most of courses involves computers. But 50% of students are willing to learn also different types of operating systems. Then they will be able to work as network administrators, which are mostly done using UNIX based systems.

VII. Students opinions and suggestions

The last part of our questionnaire wasn't question but we wanted to know what students want to say generally about programming teaching at our faculty. We divided students opinions and suggestions into following sections.

A. More or less programming?

The first group of students haven't got any programming courses because their specialization didn't offer any, but they still wanted to learn how to program.

More programming courses for non-informatics specializations because today many employers demand them.

Another answers leaded to courses of economy. Students suggested that they have to teach too many of them. They would be replaced by more useful programming courses for their praxis. On the other side some student doesn't like to program and think that we should be focused on other engineering courses.

There is too much informatics but less and less electronics or measurements in electrical engineering.

B. Modern courses

The next group of student wanted to modernize the programming courses. Better students can really good see what is used in praxis especially if they begin part-time work meanwhile study.

We learn old topics and a lot of courses are theoretical with no connection to praxis.

Students want more practical examples rather than theoretical lectures. Examples help students to better understand the topics and help them when students want to try what they learned.

More study materials and examples, not only presentations.

As we written in section about diversity of students some student didn't had computer programming courses at their secondary schools. Therefor these students wanted to have better basis of algorithm designs and different approach to them.

Not everyone that came from secondary school with basic knowledge of programming. Teacher should notice that students aren't often on the same level.

Few students demanded to use more open source projects as tools and operating systems during lectures. He argues about price and maintains cost and difficulty advantage for them.

Use operating systems and tools that can student use with no charge legally.

Last small group of the students wanted to learn much more about computer networks design and server management. They mentioned that this is widely used and wanted competence for the graduates.

I want to be learned much more about computer networks and server management and maintenance.

C. Connection to praxis

As we mentioned some students work in companies during study. They see what they do and what we teach. Therefor they want more tools, which are necessary in praxis. Most suggested tools were subversion for teamwork. Another useful technique is programming management and job scheduling. Also modern developing environments would be used during teaching process. This may be too expensive for us to buy commercial tools to teach this. Therefor we are forced to use much more open source tools with customization to our needs. Lecturers alone can do this customization.

I would like to have basics of teamwork on projects with SVN or Eclipse and Subclipse.

Another demand was to connect lectures to their courses in their specialization or some visits to real-world companies. It is not enough companies that allow us to come to their offices and manufacturing buildings to show students their technologies.

It would be good if as part of practice we would go on excursion to some company and see what it works there what we learn at the faculty.

D. Evaluation

Small group of students like to change evaluation of task. Very surprising were statements that we would take more of originality check of student programs made for classification. Some students copy part of program from another student to achieve more points. Group of students wants better control for originality.

Actively prevent plagiary.

Experienced students are often bored on courses for beginners and they demanded harder tasks. It is too hard for less prepared students to solve harder tasks. We also cannot divide students to the groups with similar level of knowledge because students are to choose time of their practices individually.

More tasks, but with better consulting.

E. Teachers and students personality

Students are very sensitive to teacher's personality. They usually see if we are in good or bad mood and how we behave to them. In question about what programming courses did student problem they answered also this

Problem isn't anything if student have kind teacher.

One student wants that we could learn soft skill like writing mail or communication with others. But this is not case to teach at computer programming courses.

VIII. Conclusion

In this paper we written about how could students affects our improvement of some university courses. The easiest way is to ask them through small questionnaires. Student want to improve what they learned and our questionnaire was proving it while one fifth of all faculty students answered.

For most problems in first few years is responsible diversity of the students. Student came to universities form different types of secondary schools where different types of teachers learning different types of computer programming languages. At some secondary schools student even hadn't any programming courses at all. Few students learned some programming language by them self.

We must alter our courses to decrease differences between students especially during first few terms. The flipped Gaussian like distribution is proving that student can be divided into two general groups. One group is extremely easy to teach and other is extremely difficult to teach for us. This can bee seen also on the popularity of the courses when almost half of the students found programming course easy and other half found it very hard.

We can alter our courses based of this fact that we approach to experienced and less prepared students separately. On the same course experienced students could have harder advanced topics that are suitable for them and then we could work more with worse prepared students.

We must also consider new interesting courses for student that we don't learn often but are extensively used in praxis like programming and informatics law, teamwork or versioning and a revision control. Finally we must listen what students want and incorporate as they have good suggestions for our courses.

References

[1] J. Kinast, Ch. Reiermann, M. Sauga, Labor Paradox in Germany: Where Have the Skilled Workers Gone? [online], *Spiegel online*, SPIEGELnet GmbH 2007 cit: 11.05.2011 available at <http://www.spiegel.de/international/business/0,1518,490031,00.html>

[2] C. Barry, Germany Needs 34,000 Engineers, *Product Design & Developlemnt* [online] Advantage Business Media, 2010, cit: 11.05.2011, available at <http://www.pddnet.com/news-germany-needs-34000-engineer-112410/>

[3] AFP, Westerwelle: Germany needs foreign workers, *The Local* [online], The Local Europe GmbH, 2010, cit: 11.05.2011, available at <http://www.thelocal.de/national/20100804-28957.html>

[4] A. Zaharim, Y. M. Yusoff, M. Z. Omar, A. Mohamed, N. Muhamad, Engineering Employability Skills Required By Employers In Asia, *Proceedings of the 6th WSEAS International Conference on ENGINEERING EDUCATION*, Rodos, Greece, ISBN: 978-960-474-100-7

[5] N. Khamis, S. Idris, Issues and Solutions in Assessing Object-oriented Programming Skills in the Core Education of Computer Science and Information Technology, *12th WSEAS International Conference on COMPUTERS*, Heraklion, Greece, July 23-25, 2008, ISBN: 978-960-6766-85-5

[6] S. Garner, The Cloze Procedure and the Learning of Programming, *8th WSEAS International Conference on COMPUTERS*, Athens, Greece, 2004

[7] S. Dehnadi, R. Bornat, The camel has two humps (working title) [online], Middlesex University, UK, 2006 cit:11.05.2011 available at <http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf>

[8] C. J. Costa, M. Aparicio, R. Pierce, Evaluating Information Sources for Computer Programming Learning and Problem Solving, *Proceedings of the 9th WSEAS International Conference on APPLIED COMPUTER SCIENCE*, 2009, pp. 218-223

[9] RADU HANZU-PAZARA, EUGEN BARSAN, Teaching techniques – modern bridges between lecturers and students, *7th WSEAS International Conference on ENGINEERING EDUCATION (EDUCATION '10)*, Corfu Island, Greece July 22-24, 2010, ISBN: 978-960-474-202-8

[10] D. T. D. Phuong, F. Harada, H. Takada, H. Shimakawa, Collaborative Learning Environment with Convincing Opinions for Novice Programmers, *5th WSEAS / IASME International Conference on ENGINEERING EDUCATION (EE'08),* Heraklion, Greece, July 22-24, 2008

[11] J.A. Marin-Garcia, J. L. MAURI, Teamwork with University Engineering Students. Group Process Assessment Tool, *Proceedings of the 3rd WSEAS/IASME International Conference on Educational Technologies*, Arcachon, France, October 13-15, 2007, pp. 391 – 396

[12] J. A. Betancur, C. Rodríguez, I. Ezparragoza, An undergraduate collaborative design experience among institutions in the Americas, *Proceedings of the 8th WSEAS International Conference on Engineering Education (EDUCATION '11), Proceedings of the 2nd International Conference on Education and Educational Technologies 2011 (WORLD-EDU '11),* Corfu Island, Greece July 14-16, 2011, ISBN: 978-1-61804-021-3

[13] J. A. Marin-Garcia, J. L. Mauri, Teamwork with University Engineering Students. Group Process Assessment Tool, *Proceedings of the 3rd WSEAS/IASME International Conference on Educational Technologies, Arcachon*, France, October 13-15, 2007

[14] D. Rigas, K. Ayad, Guidelines for Edutainment in E-learning Systems, *10th WSEAS International Conference on SOFTWARE ENGINEERING, PARALLEL and DISTRIBUTED SYSTEMS (SEPADS '11*), Cambridge, UK February 20-22, 2011, ISBN: 978-960-474-277-6

[15] P. Pocatilu, F. Alecu, M. Vetrici, Using Cloud Computing for E-learning Systems, *Proceedings of the 8th WSEAS International Conference on DATA NETWORKS, COMMUNICATIONS, COMPUTERS (DNCOCO '09),* Morgan State University, Baltimore, USA November 7-9, 2009, ISBN: 978-960-474-134-2

[16] S. C. Cismas, Questionnaire for Implementing Open Distance Learning for English in Engineering, *Proceedings of the 9th WSEAS International Conference on DISTANCE LEARNING and WEB ENGINEERING*, Budapest Tech, Hungary September 3-5, 2009, ISBN: 978-960-474-115-1

[17] A. Trifonova, E. Georgieva, M. Ronchetti, Determining Students' Readiness for Mobile Learning, *Proceedings of the 5th WSEAS International Conference on E-ACTIVITIES*, Venice, Italy, November 20-22, 2006,