INTRODUCING COMPUTER SCIENCE INTO CZECH GRAMMAR SCHOOLS: FIRST RESULTS

Daniel Lessner

Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University in Prague (CZECH REPUBLIC)
lessner@ksvi.mff.cuni.cz

Abstract

In this paper we share the first findings from teaching computer science (CS) on Czech grammar schools (GS). We first describe the basic situation and the goals of our research. Then we describe briefly the goals, content and methods of the developed CS course. Afterward we proceed to the first results, as we are at the end of the first school year of experimental education.

Czech GS (students aged 15–19) aim to provide classical education and general knowledge as a basis for further university studies of nearly any field. However, CS is not a part of standard grammar school education in the Czech Republic. It is mentioned only briefly in curricular documents, and often completely missing in real classes (unlike information technology related education). We see this as a mistake, considering the role of CS in our lives and its potential to develop higher cognitive skills and key competencies (mostly problem and communication related).

As a part of our effort to change this, we have developed an introductory computer science course. We have determined the goals and established the necessary core of the subject, both in relation to other GS subjects. Then we needed a teaching methodology, as the goals and topics may be rather uncommon. This basic combination of content and methods is to be tested in practice and evaluated against the original goals. Acquired results can then serve as a base for thinking of introducing CS into our GS system-wide.

CS on GS shall widen general knowledge and improve cognitive skills for everyone, not only future computer scientists. We also need to stay in accord and connection with other GS subjects. Traditional approaches to basic CS topics must therefore be reworked. We have created a CS program proposal for one school year, 90 minutes a week. It covers the chosen main topics and concepts, which include information, graphs, problems, state space, algorithm, recursion, efficiency, Turing test and determinism.

Tested teaching methods are based strongly on didactics of mathematics, which is perhaps the closest GS subject to CS. However, CS seems to be more abstract. We need to work with plausible sample situations and allow students to actually experience them themselves. When it is possible, we prefer constructivist approaches. It is time consuming, thus requires extra care when distributing content into the program.

The outcomes of the first year of CS teaching serve as a valuable feedback. We have now an idea about the limits of CS education in the given context (including time frame, method and knowledge from other subjects). The main results are as follows. Basics of CS are fairly well accessible for GS students and motivating and beneficial at the same time. Most of the goals we have set for the course is actually achievable. The chosen methods work as expected. These results are used to refine the course for the next school year and to allow us to obtain more detailed findings about CS education effects.

Keywords: Secondary education, grammar schools, curriculum design, computer science.

Acknowledgment: The author thanks his supervisor RNDr. Tomáš Holan, Ph.D., for guidance during research and for assistance with editing parts of this paper.

1 INTRODUCTION

The main document specifying education on grammar schools[1] (GS) in Czech Republic is the Framework Education Programme for Secondary General Education (FEP, [1]). Every school

[1]Secondary education, students aged 15–19, years 10–13 from entering compulsory education.

creates its own detailed programme based on FEP. It states that GS aim to provide classical education and general knowledge as a basis for further university studies of nearly any field. This includes developing so called key competences [2], which among others include problem solving and communication skills[2]. They stand above classical subjects, which are mainly means to develop these key competences.

However, computer science (CS) is not a part of education in Czech grammar schools. FEP mentions CS only briefly and rather ambiguously. CS is consequently missing also in actual classes. There is a compulsory area focused on using digital technology, but no systematic approach to computer science meaning efficient information processing in general [3, 4]. Most of grammar school graduates are not even aware of its existence.

We believe that computer science should be included in grammar schools as a subject equal to other sciences. The reasons are as follows. CS seems to be very promising in developing problem solving and communication skills, which happen to be among main goals of secondary education. Further, we consider CS to be a part of general knowledge, due to its influences in other areas and everyday life. The strong bond between CS and mathematics could help stop the bad trend in Czech students' mathematical literacy [5]. More reasons and more detailed discussions are given in [6–9].

Our research aims to help introduce CS into our grammar schools. The first phase was to examine the situation, both in Czechia and abroad. As it turned out, they are quite ahead in many countries [10–14]. Next step was to specify what CS for GS should actually be and to design a pilot course. Then we tested this course with GS students. The objective was to find the limits of such topic on grammar schools and to examine the suitability of chosen methods and content.

We are getting to the end of the first year of this testing now. This paper describes the course, the experimental teaching and summarizes our main findings. We will use these findings to refine the course. After further testing and evaluating we will publish our results as an input to the discussion about including CS into our GS system-wide.

During the first year of testing we have already collected results, as we believe, interesting and valuable enough to be shared in this paper. We first describe the design of the course including its goals, structure, content and basic methodology. Then we describe the actual context which we worked in. All that is a preparation for the core of this article, which lays in observations made during the course.

2 THE COURSE DESIGN

In this section we describe the proposed and tested computer science course to the extent necessary for understanding the section with observations and results. The course was not developed by adjusting university courses, as they have different goals, or by adjusting foreign GS programmes, as they differ in context. Details on the course design can be found in [8].

2.1 Goals

General goals of computer science education have to be based on and in compliance with the general grammar school education goals. That means developing key competences (we focus mainly on problem solving, communication and studying skills) and providing basic overview over computer science. Students shall become familiar with its subject, methods, connections to other areas, fundamental results and their implications and applications.

Students shall also be able to actually use the basics of CS in everyday life. The point is not to use the exact algorithms. Students should acquire basic habits and use appropriate heuristics based on some solid theoretical knowledge and practical experience. They should not only *be* efficient, they should be also *aware* of that they are.

Not to be forgotten, key competences include also stances and opinions which are to be cultivated. Examples to focus on in CS are preferring systematic and efficient work, thinking before acting and feeling the urge to leave repetitive tasks to machines.

The CS course aims to teach information technology use no more than any other GS subject: digital literacy is being developed while using the technology, sometimes in extraordinary situations.

[2]All six key competences defined by FEP are: problem solving, communication, learning, civic competence, entrepreneurship and personal and social competence.

However, this happens by the way and is not the purpose of CS course. Similarly, the CS course does not aim to teach programming. There are eligible seminars for that. It is certainly easier to teach CS to someone with at least basic programming skills. However, we have concluded that it is more efficient to use the given time for teaching CS directly instead of teaching basics of programming first.

More specific objectives are mentioned in the following subsection.

2.2 Structure and content

We have determined the structure according to usual conditions on grammar schools. The course is designed for one school year consisting of ten months. The usual time frame for this type of subjects is two teaching hours weekly, often joint to make a 90 minutes lasting lesson. We split the course into modules, each supposed to last approximately a month (i.e. four lessons, six complete hours in total, with some extra time in home assignments). Each one of these modules deals explicitly with one fundamental idea, while other important areas are in the background.

These fundamental ideas are based on the above described goals. They therefore differ from the classical content of a usual introductory CS course on universities. We had to consider students' maturity and difficulty of other grammar school subjects. The main ideas and illustrating problems must be comprehensible, but challenging, and in the core of CS, but useful also outside of it.

The main modules are therefore about information, graphs[3], problems, algorithms and efficiency. The course was supposed to begin with an introductory module to review some necessary mathematics, but we have decided to blend it into the other modules, as described below. The plan involved three more modules: a review using topological sorting, an introduction into an advanced CS topic, such as evolutionary algorithms, and connections to humanities, discussing topics such as Turing test and free will and their impact on the society. As it turns out, we will have to omit two of these modules due to time reasons described further in this paper.

The module *Information* is the true introduction into the course, as the Czech translation of computer science is based on the word information. Students shall learn what it is, how do we measure it and store it, what the consequences of those ideas are and what the related terms are. The concept of decision trees is introduced. It serves also as one of the first prototypes of an algorithm and allows also some thoughts on efficiency and optimization. This module requires numeral systems (especially the binary) and logarithms review. Students shall be able to determine the informational value of a statement and to ask a question promising the maximal informational gain.

The module *Graphs* is to show students this extremely useful structure, its main characteristics and possible applications. Graph notations are included to show to students that even such a visual structure can be encoded in a character based message, and to give them a tool to handle large and possibly confusing graphs. The main problem to solve during this module is to distinguish Eulerian graphs[4] and related issues. It is motivating, useful and simple enough at the same time. The ultimate topic to discuss is isomorphism of graphs.

The module *Problems* aims to help students to deal with problems in general. This requires clarifying of terms and some general heuristics. They choose their opinion on the term solution: Is it solely the information on how to solve a problem, or is the actual carrying out of the solution from the first case? Or in other words, do they consider a problem solved by seeing a way to do it? Students shall understand the necessity of defining the initial and desired final state. The space between is then filled with the concept of state space, which serves also as a rough tool to assess the difficulty of a given problem. Students solve several puzzles and other problematic tasks, searching for generally useful heuristics. An important algorithm studied in this module is backtracking and its universal use together with state space. Motivated students study also two player game spaces, as a modification of state spaces. At the end of the module, limitations of state space applications have to be revealed.

The module *Algorithms* is an introduction to this fundamental notion. Students have to understand the meaning (e.g. reliability, no reasoning, usefulness) and the resulting characteristics defining algorithms. Using the definition, students shall be able to tell apart algorithmic processes. They also get to know the most common non-algorithmic instructions. The concept of decreasing measure to proof finiteness is introduced as an example of advanced CS tool. All this is supported by examples

[3]Not graphs of functions, but graphs as sets of vertices and edges to denote binary relations.

[4]Eulerian graphs are drawable using only one uninterrupted stroke.

from previous models as well as a few simple new ones according to students' interests. The limits of algorithms are shown: there are well defined problems which are unsolvable by principle. Self-reference and infinity are the main traits to be aware of.

The module *Efficiency* allows students to reasonably compare algorithms. It includes the very basics of classical complexity theory together with some connections to practice – an optimal algorithm is useless, if it takes too long to implement. Processing sorted lists is the main source of examples. The module would be of course incomplete without some basic optimization heuristics. Finally, students should understand that exponential algorithms, although feasible in theory, are practically useless.

The remaining modules have not been tested yet and are unimportant for understanding the following sections. We will therefore omit their detailed description.

2.3 Method

As CS is not present on our grammar schools, there is also no systematic teaching methodology. Here we describe briefly the assumptions on which we built the main traits of the resulting methodology. Educational goals are of course the first to be considered while assembling and developing such methodology. As we seek true understanding and ability to apply the matter in new situations, we prefer constructivist approach where possible.

Students are to work with realistic sample situations in an effort to solve them themselves. We do not train students to apply routine procedures mechanically[5]. They are to find and use the necessary tools for solving a problem themselves. This is what they spend most of the time with. We could easily tell them more in shorter time, but the effect would be very likely only temporary and shallow.

The tools students can use are not limited. Computers are rarely necessary, but available. Most of the students are actually rather distracted or otherwise stalled by them. However, students have to decide themselves, when is it worth to use a computer for the given purpose.

The closest subject to CS on grammar schools is mathematics. This is why the teaching methods are strongly inspired by didactics of mathematics. However, it is important to be aware of the distinctions. Structures and concepts of grammar school mathematics are less abstract and they can be applied mechanically in most cases. On the other hand, direct practical applications are in some topics rather artificial, if even existing. Trigonometric equations could serve as an example. Concepts included in CS are more difficult to understand, but have sometimes more straightforward and plausible motivations. CS models are more practical for describing various processes. Answers in CS are not only correct or wrong. They are of different efficiency according to different criteria. Another distinction against the usual mathematics education is that we avoid formal definitions until they are needed for precise communication or other important reason.

Ideas like algorithm, complexity and others are very abstract. That is why we intentionally work with them long before they are defined explicitly. This also explains the order of modules, which might be considered unusual. Algorithm comes only in the half of the course, as if we would not know how a fundamental CS notion it is. The reason is that students already have a sufficient number of well-known examples and enough experience when they are exposed to the term itself. It is therefore not as abstract and empty for them as if we would come up with it in the beginning of the course. This principle is used with other deep notions too.

We try to let students work not only with their imagination, but also using actual data and physical objects. This is helpful for those with worse abstract thinking capabilities. It also allows us to use natural rewards and punishments. Efficiency is not only a result on a paper approved by the teacher, it actually saves time and effort. Another way of motivating students to efficient behavior is including various games or contests into the course.

We use home assignments to save time. They sometimes involve exercises of basic skills, sometimes problematic tasks which need to be thought through during longer time period and do not need the teacher at hand. The results usually serve as a starting point for the subsequent lesson.

Above described principles has led us to a general scheme, which more or less applies to every module. First we tackle with the motivating problem (or more of them). Solving it eventually leads

[5]We actually do, but on a higher level. They are not to apply a specific routine to solve any special kind of problem. They are to employ general heuristics a computer scientist uses to solve problems of any kind, e.g. as in [15].

to a more general model and method to solve other problems of that kind. It also usually turns out that this “kind” of problems is much wider than expected. Other examples serve for further practice and examination of the theory.

The last part included in the usual module scheme is some information about the limits of the method. Applicability of state space may serve as an example. It is a useful concept for solving problems. But not applicable well when it is too big, without full knowledge about the system, with chance involved or when it is changing continuously. We consider it fair towards the students to inform them, before they run into some needless trouble[6].

Limits are also often fundamental results of computer science itself which are simply worth being aware of as a part of general knowledge. The existence of well defined, yet computationally unsolvable problems is an example, practical unsolvability of NP complete problems is another one [3]. Some of these theoretical limits are not reachable in a constructivist manner.

2.4 Feedback

Students receive most the feedback from their peers, as they often work in groups. This also makes them express their ideas in a comprehensible way, what is a challenge for most of them as soon as the idea is not trivial. The teacher provides some more informal feedback while consulting the groups in trouble.

The teacher also comments the home assignments. Moreover, an early submission gives a chance to resubmit the assignment based on the teachers comments. This motivates some students (not all) to stay on their schedule. Formal assessment, which is also a base for final grading, comes at the end of each module in the form of a written final test.

Students find these tests unusually difficult. The matter itself is quite abstract and the tests often last the whole lesson (90 minutes). There are also aspects making them easier. Students are encouraged to use their lecture notes and usually also the internet (although any cooperation is not allowed). How they use these possibilities is of course up to them. This enhances their studying skills (taking useful notes, working with information sources efficiently). The second important consequence is that the tests can contain also some unusual tasks. Students have to show the ability to correctly apply what they have learned in new contexts.

The tests also include bonus tasks. They usually require some deeper understanding of the topic or a bright thought. Points for bonus tasks are not included in the expected total. Students can thus only win extra points by solving these tasks. Again, they themselves have to assess the situation, determine the most promising approach and choose which tasks to solve first. These tests are stimulating for some students, for some they are rather stressful – it is, however, also something to learn to deal with.

3 REALIZATION

After describing the intended course we describe the actual teaching and the relevant conditions to complete the image and thus allow the reader to better understand the below presented findings. As computer science is usually not included in grammar school programmes, it is a challenge to find a chance to work with actual students. For the first year, we worked with four eligible informatics and programming seminar groups on three different schools.

Eligible seminars take 90 minutes weekly, as expected in advance and as it is usual for eligible seminars on Czech grammar schools. They take part in the afternoon, hence the students are not very fresh anymore. Every student has a personal computer available in the classroom. Some students use their own computers.

Groups are of 8 to 12 students, there is one girl in each group. Most of the students are in their last year on GS and do not intend to continue their carrier in CS anyhow. However, some of them have chosen informatics as a subject for their school leaving exams. Most of them will be examined on mathematics, due to rather complicated examination rules[7]. Approximately half of the students have signed up for the seminars as for “the best of bad options”, because they just had to pick one. They want only to pass the subject as effortlessly as possible.

[6]It is also interesting for students to discuss the possibilities of overcoming these theoretical limits. They usually involve some creative thinking out-of-the-box. They mostly too advanced though, therefore not included in the basic course.

Students' experiences vary broadly between the groups as well as inside. Only a few had any knowledge about computer science itself. Some of them are very fond of mathematics, some are avoiding it whenever possible. Some of them have never programmed before the seminar, some of them did at school, and some are making money with it.

Our involvement in the seminars varied according to specific conditions at each school. One seminar was taught entirely by us, in one the regular teacher took part. The last two groups we were visiting approximately once in a month. We made a minimized or partial version of each module for this purpose with some follow-up home assignment. Students from the last two groups have written no final tests. Comprehension was checked orally at the end of the lesson and on the next CS lesson.

The last important aspect to mention is the double set of goals of our teaching. While pursuing our research questions, we had to accomplish other goals too, e.g. prepare students for their school-leaving exam (or at least not to occupy them too overly) or work on original seminar goals, such as programming. This double-tracking sometimes limited the research. Students' needs of course had the priority.

A major change to the course design made while adjusting it to be compatible with each school programme was inclusion of some programming. This was the case with every group. Choice of the language was up to the school or up to the students. They worked with Python, Pascal, C++, C# and PHP. We encouraged them not to specialize on one language and to get a taste of more of them. Again, the goals of education on grammar schools are some general knowledge and key competences. Mastering a programming language is not so important. It is therefore more beneficial to meet and work with more formal languages than with one.

4 OBSERVATIONS AND RESULTS

Here we share the outcomes of the first year of CS teaching. They consist mostly of individual findings related to each topic. They are based on observations during lessons, discussions with students and their teachers, assignments and tests. First we describe general observations, which have appeared in multiple modules and of course in multiple groups. Then we describe a few topic specific findings.

Described groups are not to be compared, as they work in systematically different conditions. However, we have obtained some common results, as described below (as well as some very different). The aim of the experimental teaching was to confirm the fundamental suitability of cornerstones of the proposed CS course, basic parameters of the course and the usability of individual educational activities. The qualitative findings are therefore of higher importance than test scores.

4.1 General findings

One of the major tasks when preparing the course contents was to find good motivating problems. Results of this are rather mixed. Some groups have reacted very well, some reacted well to an alternative problem, some seemed to be rather untouched anyway. Variance of excitement inside the groups was very little, there was always a strong major attitude. Motivation of course is not based solely on the discussed matter. We hoped to overcome the basic conditions in some groups (i.e. CS not a subject of any interest or importance), but we did not succeed in some cases. These students were often determined about subjects of their interests (e.g. chosen school-leaving exam subjects) and in all other subjects did only enough to pass. This will lead to some optimizations in course content and feedback system. However, considering CS being quite deep and abstract and out of the official curriculum, the course was accepted very well.

Another major question was whether the content is even comprehensible to grammar school students. As it turns out, it is[8]. Most of the basic knowledge and skills (e.g. the concept of binary search) were acquired. This was confirmed also by final tests after each module. We decided not to use any

[7]In the current initial period of state-wide uniform exams, one of the choices to make is between mathematics and a foreign language. This is supposed to be temporary. The intended final structure of the uniform exam includes Czech language, a chosen foreign language and one of the following three: mathematics, humanities or informatics. Informatics is there a mixture between use of digital technology, information science and computer science. The rest of the exam is not unified and can differ to some extent on each school.

[8]However, the reader is to be reminded that we worked with attendants of eligible seminars, hence this is not to be generalized directly to the whole grammar school students population.

pretests, because the students mostly did not even understand the questions asked. We have also tried to go beyond the preset goals. Interestingly, it led to no success. The goals have been set well for the given conditions. More details on the limits of included content are given in the next section.

Our progress was slower than expected. The original plan was intentionally fuller, in case we would proceed faster – we did not want to add topics ad hoc. It included ten modules, one per month. As it seems, we will end with the seventh. The main reasons are of course holidays and special events (e.g. various field trips), school leaving exams for some students (it shortens the school year by two months), programming inclusion, and of course the teaching methodology itself, which is rather slow by its nature. However, the most important goals have been met. Students do not consider the time lost or wasted inefficiently. Moreover, students were actively using or recognizing older matter even after months without being reminded. Hence we also do not consider the extra time taken to explore a topic wasted.

Our way of teaching was troublesome and confusing for the students at first. We did not want only the results, we wanted them also explained precisely. This is something students were not used to. Another source of confusion was the liberty in methods they were allowed to use. Students have not been provided with explicit and detailed instructions right away.

These difficulties have turned out fully during the first two final tests. Students had no problems at all with questions requiring mechanically applied routines (e.g. conversions between numeral systems or calculating the efficiency of a decision tree). They often stick to their routines, even when they were not efficient and a little thought would allow them to proceed much faster. Students preferred certainty. However, after eight months of collaborating, they understood our requirements well and they also found the tests more feasible. The “ask more, get more” principle worked well in our situation.

Some of the ideas encountered in the first module were needed again during the school year. More than a half of students (in both groups) recalled them actively. This included bisection, decision trees, binary numeral system and efficiency modeled using some kind of elementary step.

An important issue encountered many times during the course is lack of mathematical skills. Even in their last year, students do not understand some important topics enough to use them. The main among these topics are logarithms and exponential functions, sets, the concept of probability and basic combinatorics. We have therefore dealt with these topics in our lessons when needed, even though we could not have it done very thoroughly. It can be interpreted also in a positive way: even without prior knowledge, computer science is comprehensible and useful. Moreover, as the situation made students need that knowledge, they were quite motivated to master the needed skills.

Students' communication skills were rather disappointing. Lot of the terms (e.g. information) seem to be too abstract to talk about comprehensibly, even though students solve the related problem correctly. Graphs turned out to be a good topic to help students realize their deficits and then to try to diminish it. While talking about graphs, they have to make clear whether they mean vertices or edges at the moment, and they of course have to mind quantifiers very carefully.

Another tricky area is determining whether some procedure is an algorithm. They shall not use the term until the process is proven to be an algorithm. When talking about efficiency of algorithms they must at least specify the case – expected, average, best, worse – and the criteria to consider efficiency (misunderstanding in this was perhaps the most frequent source of disputes on efficiency among GS students).

We can happily state that we have witnessed an obvious improvement in students' communication skills by the end of the courses. The reason is perhaps in the necessity to express themselves during group activities, in home assignments etc., using also natural language to describe algorithms.

The inclusion of programming into the course had the expected advantages and disadvantages. On one hand it was slowing down the computer science program, on the other it helped students to understand a few algorithms. The aim was not to teach students how to transform their ideas into a programming language, that would be much more difficult and time consuming. Programs served rather as a description of algorithms, as a mean of communication, and as a way of exploring how the algorithms behave on larger data. Students were tweaking them and adjusting rather than creating.

Still, it was too much at once for some of the students. It is clear that in the limited time we have to suffice without programming, when teaching usual grammar school students with no extra interest in computers.

4.2 Module specific findings

Here we discuss further observations, which are linked directly to individual modules. Described limits of students' comprehension and other statements must be of course considered in the given context and cannot be generalized.

The first module in the course deals with *Information*. We have decided during the first lessons to blend the preparative (mostly mathematical) topics into the other modules and to work through them only when needed. This way we got to the real matter sooner and students were more motivated to understand the necessary mathematics, facing some specific problem. There are of course limitations. Students' understanding of probability for example is far too shallow to understand the term information based on entropy. We could eventually get there, but the time invested would not be worth it regarding our goals.

The module *Graphs* turned out to be very fruitful regarding small hypotheses to be expressed precisely and checked. The limit of students abstract imagination lays slightly behind planarity. They all understood after some time that one graph can be drawn in many ways and some, yet not all, may be planar. Isomorphism on the other hand seemed to be too much for most of them, as expected. Interestingly, they were convinced they do understand the idea well enough. However, their solutions of the given tasks showed us otherwise.

An interesting situation developed regarding terminology. In adequate cases, we provided students with the usual term for the discussed notion. Afterward, when someone used the terms incorrectly and expressed him or herself ambiguously, a noticeable social pressure arose to put the expression right.

The main concept introduced in the module *Problems* is state space. This was rather a failure. Students did not understand it in its general form. They have stuck to the first or second example (measuring some amount of water with bowls of given volumes), unable to rise above and see the abstraction. This module is to be reworked. We have to find better examples. We will also try to let students experience some examples beforehand in the previous modules.

Another important goal was to make them realize general heuristics they use to solve problems. They were indeed interested in identifying them and using them willingly in new situations. However, we have not managed to turn this into a systematic habit of any kind. We believe this would require more time. Possibly influenced by our encouragement, some of the students stayed interested and were aware of their strategies during upcoming modules too.

In the module *Algorithms* an interesting misconception appeared: including correctness (sometimes even efficiency) into the definition of algorithm. This would make the "algorithmicity" of a process dependent on specific tasks. Another interesting observation is that in three groups, students did not see any sense in focusing on processes, which can be carried out by machines. They were completely unaware of the extent to which machines algorithmically influence their lives and the society itself, although it should have been included in their previous education.

While writing this contribution, we are in the middle of the last core module, *Efficiency*. After some experience with algorithmic processes, we arrived to compare them and to find ways how to optimize them. The proposed problems are intentionally very open. We can see on what are students focused, and at what level are they able to think about efficiency. In theory, they are even capable of understanding the basic concepts of computational complexity and find them reasonable and useful. However, actual calculations do not seem to be possible at this level.

Students do slightly better with actual data in their hands. They are able to estimate an approximate number of steps to complete an algorithm on an input of given size, but they have difficulties to express the relation using variables or to compare such relations. It is the same with constructing the algorithms themselves. They work much better with sample data than with a hypothetical general situation. This surprised us, regarding their age (ca. 17–20). We believe that this would be different, if they had to deal with algorithms also earlier during their studies.

Even though we believed the proposed tasks would be trivial, students have found many fruitful areas. Instead of simply sort two lists and compare them, their thoughts ran to topics as complex as stacks, trees and tries, traveling salesman problem, greedy algorithms and alike. This is due to the "realistic" background of the problems, such as mail delivery, shopping or library stock-taking. Students do not find the abstraction we would and therefore solve a slightly different problem. This is a very beneficial turn out and we will keep it on mind when adjusting the problem settings for next year.

Observing students' effort and the suboptimal solutions they come up with, we conclude that the *Efficiency* module is very necessary for them. After certain amount of work on optimizing algorithms, they are able to generalize their findings, such as "use what is already done, do not reinvent the wheel", "use the auxiliary results, do not calculate anything twice", "stop as soon as you know the answer, do only what you have to", "work in parallel, when possible". This is perhaps the most important goal of this module.

Above listed findings serve us as a priceless feedback. We have now an idea about the possibilities and limits of CS education in the given context (including time frame, method and knowledge from other subjects).

5 CONCLUSION

Computer science is not a standard part of Czech grammar school education. We described here the concept of an introductory CS course, the context of its experimental realization and the results we have obtained after almost one school year.

Summarizing all the experiences gained so far, we may conclude: teaching CS on Czech GS is possible. Students are able to enjoy and master the matter and achieve the educational goals on a similar level as with other subjects. It is possible despite the fact that lot of useful mathematics is traditionally placed only into the last year of GS.

Based on these results, we can adjust the program during the summer holidays. Then during the next year we will examine higher and more general effects of CS education, including problem solving and communication skills.

REFERENCES

[1] (2007) Framework Education Programme for Secondary General Education (Grammar Schools). Výzkumný ústav pedagogický v Praze, Praha.

[2] Slejšková, E. (ed.) (2008). Klíčové kompetence na gymnáziu. Výzkumný ústav pedagogický v Praze, Praha.

[3] Aho, A. V., Ullman, J. D. (2000). Foundations of Computer Science, C Edition. Computer Science Press, New York.

[4] Vaníček, J., Papík, M., Pergl, R., Vaníček, T. (2007). Teoretické základy informatiky. Kernberg Publishing, Praha.

[5] Faltýn, J., Nemčíková, K., Zelendová, E. (eds) (2010). Gramotnosti ve vzdělávání. Výzkumný ústav pedagogický v Praze, Praha.

[6] Hromkovič, J., Steffen, B. (2011). Why teaching informatics in schools is as important as teaching mathematics and natural sciences. Proceedings of the 5th international conference on Informatics in Schools: situation, Evolution and Perspectives, ISSEP'11, Springer-Verlag, Berlin, Heidelberg, pp 21-30.

[7] Guzdial, M., Soloway, E. (2003). Computer science is more important than calculus: the challenge of living up to our potential. ACM SIGCSE Bulletin 35(2). pp. 5-8.

[8] Lessner, D. (2011). Computer science curriculum proposal for Czech grammar schools. ITAT'11 Zborník príspevkov, PONT s. r. o., Seňa, pp 99-104.

[9] Denning, P. J. (2007). Computing is a natural science. Communications of the ACM 50(7), pp. 13-18.

[10] Lessner, D. (2010). Computer Science Education on High Schools. WDS'10 Proceedings of Contributed Papers: Part I – Mathematics and Computer Sciences, Matfyzpress, Praha, pp. 110-115.

[11] Bell, T., Witten, I. H., Fellows, M. (2006). Computer Science Unplugged. Computer Science unplugged, Cantebury.

[12] Gal-Ezer, J., Harel, D. (1999). Curriculum and course syllabi for a high-school program in computer science. Computer Science Education 9(2), pp. 114-147.

[13] Tucker, A. (2006). A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee. 2nd Edition. CSTA.

[14] Grgurina, N., Tolboom, J. (2008). The first decade of informatics in Dutch high schools. Informatics in Education 7(1), pp. 55-74.

[15] Pólya, G. (1957). *How to Solve It*. Garden City, NY: Doubleday.