Software Engineering in Primary and Secondary Schools – Informatics Education is More Than Programming

Andreas Bollin, Stefan Pasterk, Peter Antonitsch, and Barbara Sabitzer
Institute of Informatics-Didactics
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria
Email: Andreas.Bollin@aau.at

***Abstract*—Software Engineering is definitely an important subject matter and it is taught all over the world: at Universities, at Colleges, and recently also at High Schools. There are international Software Engineering curricula, standards, and certificates, but there is no manifestation of Software Engineering (and related practices) in the course syllabi at primary and secondary schools. There are good reasons for it, but based on the authors' experiences gained in combining Software Engineering topics with school projects and based on discussions with teachers and curriculum designers, this paper shows that informatics education can be much more than just programming. Even more, the paper shows that it makes sense to interweave Software Engineering topics with school projects and to motivate for the most important practices related to that field.**

I. Introduction

One major objective of software engineers is to develop affordable programs that are dependable for consumers without bugs or glitches. For achieving this goal, Software Engineering education has to cover a broad spectrum of knowledge and skills software engineers will be required to apply throughout their professional life. Within a school setting (from primary to secondary schools) it is obvious that covering all the topics in depth is not feasible due to the previous knowledge of the pupils, the curricular constraints as well as due to the inherent differences between the school types. Furthermore, also the teachers are in most cases not really trained in Software Engineering.

On the other side, the term *Computational Thinking* gains importance in the educational sector, too. Wing once wrote in the ACM's viewpoints [1, p.33] that to "... reading, writing, and arithmetic, we should add computational thinking to every childs analytical ability". Under the guise of informatics, more and more schools (also in Austria) now additionally provide either digital literacy education (as part of or similar to the European Computer Driving Licence [2]) or programming courses. But, computational thinking is much more than using a digital device, using a spreadsheet program, or creating games with a programming language. In the aforementioned article, Wing refers, among others, to the following skills: problem solving in an efficient manner, system design, and understanding human behavior (just to mention some of them).

We now think that Software Engineering could be a useful bridge that helps teachers (but also teacher trainers at Universities) to foster computational thinking in schools. Aristotle postulated in his 5th axiom that "the whole is more than the sum of its parts" [3]. This observation is also supported by neurodidactic findings: "People have enormous difficulty learning when either parts or wholes are neglected" [4]. Hence, the learning brain needs the whole and the details; it requires both a big picture and paying attention to the individual parts [5]. In our case, the "big picture" would be Software Engineering, and the "individual parts" could be programming tasks or the meaningful use of technology.

The objective of this paper is now to report on our endeavor to find out more about the feasibility of the aforementioned "Software Engineering bridge". In order to do so, we organized a workshop at the eight ISSEP (International Conference on Informatics in Schools: Situation, Evolution and Perspectives) in Ljubljana [6] and tried to find out (a) how Software Engineering is perceived among active teachers and curriculum designers, and (b) how and if Software Engineering should or could be brought into the different school curricula.

The paper is structured as follows. Section II reports on projects that also use or teach Software Engineering topics in schools, and Section III answers the questions about the perception of Software Engineering at different stakeholders. Section IV deals with the question of how (and when) to introduce Software Engineering topics, and, finally, Section V sums up and reflects on the most important findings.

II. Related Projects

Quite a number of research papers deals with investigations towards Software Engineering education at the academic level, but a deeper analysis would go beyond the scope of this paper. An excellent representative example is the work of Lethbridge et al. [7], who argue that the SE community could have a significant impact on the future of the discipline by focusing its effort on improving the education of software engineers. Their paper identifies key challenges to be solved in order to improve the situation, and most of them can also be mapped to the situation of informatics teachers education. Our endeavor, however, focuses on primary and secondary schools, and a couple of projects and initiatives are summarized below.

Software Engineering is, for the greater part, missing in schools, except in very few special secondary schools (e.g. the "Academy for Software Engineering" in New York [8]), and only few secondary schools with a special orientation follow curricula including SE or software development as a subject. It convenes with the situation that, only recently in 2009, the Teacher Association Council reports on the *relatively new subject* "Engineering Education" (including SE) that "has slowly been making its way into U.S. K-12" [9].

In Europe the situation is comparable. At least in Bavaria/Germany, secondary school students learn basics of software projects and Software Engineering in grades ten and eleven as part of the subject computer science [10]. Another initiative is the master class for interested pupils at the Technische Universiteit Eindhoven (The Netherlands). They organized a three days course in Software Engineering (six times) and concluded that the success of this master class shows that it fulfills a need [11]. Finally, there are projects that focus on software development at schools. As one representative the work of Kastl et al. [12] can be mentioned. There, agile methods are sucessfully used to improve the satisfaction with software development projects at schools.

In Klagenfurt, we try a more holistic approach for introducing computational thinking at schools (at all levels) and also use Software Engineering as a vehicle for doing so. In 2014, past experiences motivated us to test the Software Engineering bridge the first time at a Vocational High School of Commerce and Tourism at two different grades (6 and 11) in the framework of an IMST project (Innovations make schools top) [13]. The results were more than promising [14] and it turned out that two other smaller projects at our institute, "COOL Informatics" and "Informatics – A Childs Play", have laid the foundations.

The project called "COOL Informatics" was developed at the University of Klagenfurt, Austria, to motivate for and to reduce or to avoid the fears of technology and especially informatics [15], [16]. The participants were able to attend workshops or visit working stations with different topics to explore the core concepts of informatics in a playful and illustrative way. Some of the learning lessons are based on approaches like "CS unplugged" [17] or "Informatik erLeben" (German for "Experiencing Informatics") [18] and others were developed during the preparation phase of the project. They included topics like encryption and decryption, codes, binary numbers, logic, networks and programming. Additionally Concepts like algorithms or modeling were used to support learning in other subjects from elementary education up to university level.

The second project, which is still going on, is called "Informatics – A Childs Play" [19]. One of its main objectives is to stimulate and raise the interest for the fields of technology and informatics at an early age. The intention is to show that informatics is much more than computer use. A further aspect of the project is to point out that a lot of the core concepts of informatics are included in the primary school curriculum and are practiced unconsciously in various subjects. The activities for the children participating in this project are continuous workshops at the schools, visits of the university and their own small software project considering the main aspects of software engineering.

III. Perception of Software Engineering

In 2014, we introduced Software Engineering the first time at a (non-informatics) Vocational High-School (8 students of 11th grade as "software engineers") and at a lower secondary school (25 pupils of 6th grade as "customers") in Carinthia/Austria [14]. During this cooperation we learned a lot about the perception of Software Engineering among the teachers and the pupils. Overall, it turned out that for the teachers and pupils

- the motivation was extremely high,
- it is possible to introduce and teach several software engineering competencies to pupils and teachers who are not focusing on computer science,
- there were only few difficulties with the programming environment, and
- communication skills improved noticeable.

For the teacher in the lower secondary school it was an interesting project that should be repeated.

Mid of 2015, we then had the chance to work together with active (informatics-) teachers at primary and secondary schools, but also school curriculum designers from various institutions and from three different countries. In total, we had 34 persons in a workshop, 16 of them being school teachers, the rest being educators at Universities but also actively involved in the creation of curricula for primary and secondary schools. In the workshop we wanted to find out if the Software Engineering bridge makes sense.

This section now summarizes our findings about what pupils, teachers, and experts think about Software Engineering, and what skills they think to be required.

A. Pupils' Perception

When confronted with a Software Engineering project (within the IMST experiment) the first time, most of the pupils of grade 11 said, that programming would be too difficult for them. They immediately equated Software Engineering with sitting in front of a computer and writing a program from scratch – something they have never done before. However, during the project the pupils were surprised about their own performance and the good evaluation of what they were doing (that also had an important effect – only observed, not measured – on their self-concept towards informatics in general later on). After the project and in a qualitative feedback round, the pupils (as their teacher) mentioned that the project was mainly about

- communication with others, and
- planning and thinking about how to do something in a structured manner,

something they were not so familiar with before the project. They also explained that the skills learned in the project would be very useful for them in other subjects. The introduction of software engineering concepts was organized as follows:

- Preparation phase. The 8 students of the vocational high school (11th grade) were the software engineers, but had no programming competencies. Therefore, a short introduction into the MIT App-Inventor [20] and some exercises were necessary.
- Requirements phase. The 11th grade visited the students of the lower secondary school (6th grade, 25 students). Teams were formed, each with 2 students of 11th grade and 4-5 students of 6th grade. After a brain-storming in the team they defined the topics and the requirements of the future applications and discussed them with the teachers of both schools.
- Implementation phase. The 11th grade programmed and tested the applications in their informatics lessons. Then, they developed a questionnaire for the evaluation of the applications.
- Testing phase. The 25 students of the 6th grade tested the applications, completed the questionnaires and evaluated the applications.

As we followed a full project development life-cycle where small learning applications had to be implemented for another class (the ”customers” at grade 6), they also mentioned the following knowledge areas related to their project:

- modeling and analysis,
- testing, and
- programming.

It is noticeable that quality was an issue for the project teams, too. They invested a lot of time with quality assurance activities (reviews, testing, and problem analysis). As we will see later in Section IV-A, 9 of 10 knowledge areas of the SEEK 2004 curriculum have been touched in the IMST project to some extent.

B. Teachers' Perception

From the feedback we got at ISSEP, we conclude that teachers think about the people behind the process of Software Engineering – what is an engineer doing, what are his or her skills? And, concerning the process itself, they have quite accurate explanations like ”everything necessary for producing a program” or ”the development of good software”. Their answers to the question about what Software Engineering is like can be summarized by the following two diametrical sentences (kept in their original phrasing):

- Methodologies to develop good software for a given goal where ”good” is defined by some techniques or formal properties.
- A type of engineering work about software that is successful on a project too big to be handled by one person alone.

The first statement focuses on the methodologies that are used during the development process of software. An important facette is the emphasized adjective ”good” which indicates that it somehow can be measured how good software is.

In the second statement, a very ”different view” on Software Engineering is mentioned. It points out that one person alone does not need Software Engineering during the development process. Looking closer at the statement, the word ”successful” might mean that it is called Software Engineering only if it works successfully in larger projects. Thus, this group tries to define Software Engineering by the terms ”software” and ”engineering” itself, indicating that there is a lack of knowledge in describing this field.

Both statements differ in several points showing that teachers have very diverse perceptions of the meaning of Software Engineering.

Besides the definition of Software Engineering, we were also interested in the skills needed for this profession. Skills like programming were mentioned first, but after some reflection time, the answers to this question were:

- divide greater tasks into smaller pieces,
- problem solving,
- computational thinking,
- collaborative work,
- thinking in system,
- estimating,
- modeling,
- programming and coding,

These results show that, at the end, the teachers connect a lot of the so called soft skills like problem solving, collaborative work, thinking in systems or estimating, to Software Engineering. It became clear that software engineering is about producing software systems, but programming is just one aspect of it.

It is also exciting to see that computational thinking is a topic of interest. As mentioned in the introduction section, the keyword computational thinking was introduced by Papert [21] and Wing [1] and they describe it as ”[...] a fundamental skill for everyone”. In the CSTA K-12 Computer Science Standards [22] the term is explained as ”[...] a problem-solving methodology that can interweave computer science with all disciplines” with the focus on ”abstraction, automation and analysis”. It seems that some of the skills that teachers relate to Software Engineering are also included in their notion of computational thinking.

C. Experts' Perception

When trying to find out what software engineering is about from an expert's point of view, one could survey software engineers and practitioners, and/or consider a University's course syllabus. Of course, there are different syllabi available that contain Software Engineering topics (as there could be different definitions from experts), including the Software Engineering Body of Knowledge (SWEBOK) [24]. This guide classifies Software Engineering into a set of 15 knowledge areas and it outlines the generally accepted topics in those areas. The available material, developed in a series of iterations over the past 15 years, passed through a rigorous series of

KA	Topics
CMP	Computing Essentials (172h): Computer Science Foundations, Construction Technologies and Tools, Formal Construction Methods
FND	Math. Engineering Fundamentals (89h): Logic, Discrete Mathematics, Statistics, Measurement and Economics
MAA	Modeling and Analysis (53h): Modeling Foundations, Specification and Validation of Requirements
DES	Software Design (45h): Concepts, Strategies, Architecture and HCI Design
VAV	Software Verification & Validation (42h): Reviews, Testing, Human-Computer Interface Testing and Evaluation, Problem Analysis
PRF	Professional Practice (35h): Group Dynamics, Psychology, Specific Communication Skills, Professionalism and Ethics
MGT	Software Management (19h): Planning, Personnel, Control and Configuration Management
QUA	Software Quality (16h): Quality Concepts, Culture, Standards and Processes
PRO	Software Process (13h): Concepts, Implementation
EVL	Software Evolution (10h): Processes and Activities

Fig. 1. The SEEK Knowledge Areas (KAs) of Software Engineering Education [23, p.21] including the suggested amount of time in hours for teaching the topic. The SEEK topics that are covered by our IMST project are shaded in gray.

review processes (with over 150 reviewers from 33 countries). The resulting document became an international standard through ISO/IEC JTC/SC7 [25], and it is now available in version 3.0.

Parallel to the development of SWEBOK, the ACM and the IEEE-CS sponsored a project to define curricula for different kinds of graduate and undergraduate degree programs in computing [26]. As of 2015, there are five curriculum volumes covering ”Computer Science”, ”Computer Engineering”, ”Information Systems”, ”Information Technology”, and ”Software Engineering”.

The sub-discipline ”Software Engineering” is commonly referred to by its chapter title ”Software Engineering Education Knowledge” (SEEK)[23]. It is structured into Knowledge Areas (KA) which are broken down into units and topics. The table in Fig. 1 summarizes these areas and provides a brief description (on a keyword level) of their content.

We now used the SEEK curriculum as our baseline, and in our discussions with the experts at ISSEP and together with the IMST experiences it turned out that the curriculum fulfils its purpose very well.

IV. Integrating Software Engineering

In a second step we wanted to find out if Software Engineering should be part of a curriculum at schools (primary or secondary) or not, and if teachers and curriculum designers think that including software engineering skills would be helpful to them.

A. Knowledge Areas

The experiences of the IMST project showed that it is possible to interweave Software Engineering topics with school projects and to motivate for the most important practices related to that field. The necessary key skills (stemming from the Software Engineering Education Knowledge chapter of the IEEE/ACM Software Engineering Body of Knowledge) have been tangled to some extent, but, what is more important, the participating teachers had the feeling that they could use some of the skills in their own (non-informatics) lectures, too.

The table in Figure 1 also shows which areas we covered in the IMST project, and the following knowledge areas turned out to be extremely important to other disciplines (as mentioned by the teachers):

- Professional Practice (PRF). Especially communication skills and group dynamics have been mentioned by the participating teachers more than once.
- Modeling and Analysis (MAA). Here, specifying (reading and writing) requirements was perceived as very helpful.
- Management (MGT). Planning and control were meant to be very important for a lot of other tasks in other disciplines.

Apart from that, the participating teachers had the feeling that there is no need to have an extra subject called ”Software Engineering” in their classes. The above mentioned key areas, however, cover a lot of abilities needed in other subjects.

B. Curriculum Design

Our experiences with the IMST project and the opinion of the ISSEP workshop participants match to a great extent. There is no need for an extra topic (or course unit) entitled Software Engineering. However, both groups (teachers and curriculum designers) think that Software Engineering could be an ideal bridge to improve teaching. Among other issues, the curriculum designer mentioned the following when asked about how to bring in Software Engineering areas into the different curricula:

- Focus on projects. Project-based pedagogy is useful (but the duration of a project might be too long).
- Merge modeling with other subjects.
- Focus on group work and on team work and design a game.

The first statement focuses on the notion of projects in schools. However, it has to be mentioned that the participants also added that such projects, to make sense, should include software production.

The second statement then, to our opinion, exemplifies the bridge to the other disciplines taught at schools. Modeling, as one area, can be used in various lectures - be that in a lecture about Fiber Craft, Biology, or Mathematics.

The last comment is interesting, as it means to create learning experiences as proposed by the ”Institute of Play” [27] in their quest-to-learn philosophy. Whereas their approach might still be discussed among educators, learning through games (not necessarily computer games) is, from a neuro-didactic perspective, favorable to other learning strategies.

We finally asked the teachers about their feelings when a topic like Software Engineering would enter their curriculum. To summarize, their comments were very positive, and they told us that

- they would expect a (positive) change in their lecturing style, and
- they perceive Software Engineering as a tool that helps them to modernize their teaching.

V. Summary and Outlook

This paper reported on our endeavor to find out more about the feasibility of a "Software Engineering bridge" to foster computational thinking and informatics education in primary and secondary schools. The results of a Software Engineering project at a Vocational High-School (with no focus on computer science) and discussions at a workshop at the eight ISSEP (International Conference on Informatics in Schools: Situation, Evolution and Perspectives) in Ljubljana showed that the term "Software Engineering" is perceived quite differently between teachers and curriculum designers.

Teachers still have a strong focus on programming (which can be found at pupils, too), but after a reflection phase, most of them see a combination of "soft" and "hard" skills, being valuable for other teaching disciplines at schools, too. Curriculum designers note that with a Software Engineering approach, projects, group and team work can be fostered, but, together with the teachers, they also do not see the need to integrate Software Engineering as an own topic in the different curricula. Picking out suitable topics from the SEEK knowledge areas is sufficient to them.

In all, it turned out that Software Engineering can be used to motivate for training a lot of different skills needed nowadays. This includes group dynamics, psychology, specific communication skills as well as logic, planning, modeling, and computational thinking as a cultural skill that helps us solving problems. Our endeavor also shows that it can get commonly accepted that informatics is more than just programming, and that it pays off to show the "whole" instead of the "parts".

The project of introducing Software Engineering in schools is just at the beginning and our approach is embedded in a long-term study with the objective to raise the acceptance of STEM-related topics. Currently, we are starting another intervention (teaching software engineering) in a primary school and we hope to be able to report more about our progress in a couple of months.

References

[1] J. M. Wing, "Computational Thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[2] "The ECDL Foundation," [Online] http://www.ecdl.com, Accessed: Nov. 11th, 2015.

[3] Aristotle and H. Bonitz, *Aristotelis Metaphysica*. BiblioBazaar, 2009.

[4] R. N. Caine and G. Caine, "12 Brain/Mind Natural Learning Principles," [Online] http://www.cainelearning.com/wp-content/uploads/2014/04/12-Brainmind-principles-expanded.pdf, Accessed: Nov. 19th, 2015.

[5] ——, "Understanding a brain-based approach to learning and teaching," *Educational Leadership*, vol. 48, no. 2, pp. 66–70, 1990.

[6] P. K. Antonitsch, A. Bollin, S. Pasterk, and B. Sabitzer, "Teaching Software Engineering in Primary and Secondary Schools," in *Proceedings of the International Conference on Informatics in Schools ISSEP 2015*. University of Ljubljana, Faculty of Computer and Information Science, September 2015, p. 70.

[7] T. C. Lethbridge, L. J. Richard, J. Diaz-Herrera, and B. J. Thompson, "Improving software practice through education: challenges and future trends," in *Future of Software Engineering, FOSE '07*. IEEE, 2007, pp. 12–28.

[8] "AFSE Homepage, The Academy For Software Engineering," [Online] http://www.afsenyc.org, Accessed: Dec. 10th, 2015.

[9] Teachers-Council, *Engineering in K-12 Education: Understanding the Status and Improving the Prospects*. National Academies Press, 2009.

[10] P. Hubwieser, "Computer Science Education in Secondary Schools – The Introduction of a New Compulsory Subject," *ACM Transactions on Computing Education (TOCE)*, vol. 12, no. 4, pp. 16:1–16:41, 2012.

[11] T. Verhoeff, "A master class software engineering for secondary education," in *Informatics Education-The Bridge between Using and Understanding Computers*. Springer, 2006, pp. 150–158.

[12] Petra Kastl, Silva März, and Ralf Romeike, "Agile Softwareentwicklung im Informatikunterricht Ein Best-Practice-Beispiel am Spiel "Pengu"," in *INFOS 2015: Informatik allgemeinbildend begreifen*, ser. GI-Edition - Lecture Notes in Informatics (LNI), Jens Gallenbacher, Ed. Bonn: Köllen, 2015, pp. –.

[13] W. Rohrer, "Imst project home," [Online] https://www.imst.ac.at/texte/index/bereich_id:8/seite_id:8, [Accessed: Dec. 22nd, 2015].

[14] A. Bollin and B. Sabitzer, "Teaching Software Engineering in Schools – On the right time to introduce Software Engineering Concepts," in *Proceedings of the 6th IEEE Global Engineering Education Conference, EDUCON 2015. Tallinn, Estonia*, March 2015, pp. 511–518.

[15] B. Sabitzer, *A Neurodidactical Approach to Cooperative and Cross-curricular Open Learning: COOL Informatics*. Habilitation Thesis. Alpen-Adria-Universität Klagenfurt, 2014.

[16] B. Sabitzer and S. Pasterk, "Cool Informatics: A New Approach to Computer Science and Cross-Curricular Learning," in *Proceedings of the European Conference on Technology in the Classroom 2014, Brighton, United Kingdom*, 2014, pp. 149–160.

[17] T. Bell, J. Alexander, I. Freeman, and M. Grimley, "Computer Science Unplugged: School Students Doing Real Computing Without Computers," *New Zealand Journal of Applied Computing and Information Technology*, vol. 13, no. 1, pp. 20–29, 2009.

[18] R. T. Mittermeir, E. Bischof, and K. Hodnigg, "Showing core-concepts of informatics to kids and their teachers," in *Proceedings of the 4th International Conference on Informatics in Schools: Situation, Evolution and Perspectives, ISSEP 2010*. Springer, 2010, pp. 143–154.

[19] B. Sabitzer, S. Pasterk, and E. Reci, "Informatics - A Childs Play?!" in *Proceedings of the International Conference on Education and New Learning Technologies, EDULEARN*. StudienVerlag, 2014, pp. 1081–1090.

[20] MIT, "App-inventor," [Online] http://appinventor.mit.edu/, [Accessed: Dec. 20th, 2015].

[21] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. Prentice Hall, Harvester Wheatsheaf, 1982.

[22] A. Tucker, D. Seehorn, stephen Carey, D. Moix, B. Fuschetto, I. Lee, D. O'Grady-Cuniff, C. Stephenson, and A. Verno, "CSTA K-12 Computer Science Standards, Revised 2011," CSTA Standards Task Force, ACM Inc., 2011.

[23] ACM and IEEE, "Software Engineering 2004 – Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," [Online] http://sites.computer.org/ccse, [Accessed: Nov. 11th, 2015].

[24] P. Bourque and R. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0. [Online] www.swebok.org*. IEEE Computer Society, 2014.

[25] IEEE, "Guide to the Software Engineering Body of Knowledge – SWEBOK, Technical Report, ISO/IEC 19759:2005," Software Engineering, Tech. Rep., 2005.

[26] I. J. T. F. on Computing, "Computing curricula 2005, the overview," [Online] http://www.acm.org/education/curricula-recommendations, [Accessed: Nov. 10th, 2015].

[27] "Institute of Play," [Online] http://www.instituteofplay.org, [Accessed: Dec. 22nd, 2015].