Teaching Software Engineering in Schools

On the right time to introduce Software Engineering Concepts

Andreas Bollin
Institut für Informatik-Systeme
Alpen-Adria Universität Klagenfurt
Klagenfurt, Austria
andreas.bollin@aau.at

Barbara Sabitzer
Institut für Informatikdidaktik
Alpen-Adria Universität Klagenfurt
Klagenfurt, Austria
barbara.sabitzer@aau.at

***Abstract*—Software is everywhere – be it in mobile phones, in washing machines, or in cars. With it, the importance of Software Engineering is uncontested, and Software Engineering (SE) 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. This contribution raises the question about the ideal time to start with Software Engineering at schools and reports on some first answer and lessons learned of an experiment introducing Software Engineering principles in the 3rd grade of a vocational high school (higher secondary school).**

Keywords—Engineering Education, Secondary Schools, Learning and Teaching Experiences Introduction

I. INTRODUCTION

The main goal of software engineers is to develop programs that are affordable and dependable for consumers without bugs or glitches. In order to do so, Software Engineering (SE) education must account for a broad spectrum of knowledge and skills software engineers will be required to apply throughout their professional life. Covering all the topics in depth within a school setting (from primary to secondary schools) seems to be infeasible due to the previous knowledge of the pupils, the curricular constraints as well as due to the inherent differences between the school types. Similar arguments hold for the teachers, as most of them are not really trained in SE.

This paper now shows that it is possible to interweave SE topics with school projects and to motivate for the most important practices related to that field. Key skills and challenges are identified, mapped to the situation at hand, and, by following a stepwise approach, an example setting is discussed. Based on this, the authors report their experience gained in using such an approach in a vocational high school for commerce and tourism (11th grade) in cooperation with a lower secondary school (6th grade). It turned out that, by customization of the approach, we were able to address pupils with different maturity levels, educational aims, and backgrounds.

The paper is structured as follows: firstly, it summarizes related work and describes the current situation of SE in academics and at schools (with a special focus on Austrian schools). Then, it reflects on key challenges and the necessary skills – from the perspectives of international SE Curricula – and presents the implications of including SE education in a course syllabus. Based on this, it then reports on the project "App-Programming" (which took part in the framework of the IMST project, IMST – "Innovations make Schools Top" [1]) as an example of introducing SE in a vocational high school of commerce and tourism. Finally, we further present the evaluation results of this project, reflect on them and discuss suggestions as well as implications for a possible integration of SE in (in our case) secondary education.

II. RELATED WORK

A high number of research papers deals with investigations towards SE education from many different points of view. Our project focuses on primary and secondary schools, however, to the best of our knowledge, there are no papers regarding SE and primary schools. As this paper also reports on experiences with 6th and 11th grade students only, we put out focus on secondary schools, too. Still, an exhaustive analysis is out of the scope of this paper. Therefore, this section provides an overview of selected literature and briefly characterizes their relationship in terms of three facets: (a) the role of academic SE and computer science education, (b) the role of SE in primary and secondary schools, and (c) ways of improving software practice through proper education.

A. Academic Software Engineering Education

According to the first facet, the work of Meyer [2, 3] deals with the issue of what software professionals have to know in order to efficiently achieve the goals of the different (SE) curricula. The same author focuses on teaching practices by proper identification and classification on such units as testable, reusable units of cognition, which serve for better understanding of SE topics [4]. The paper of A. Pyster et al. [5] surveys 28 SE curricula so that it gets possible to establish a baseline for graduate education. P. Robillard [6] takes a closer look at human behavior in SE that can identify ways to improve practices where creativity is involved.

Positioning of SE activities within a holistic view of the software process helps us in better defining what is specific to software engineering and to find out how to quantitatively and qualitatively measure these activities. Gregory W. Hislop [7], for example, provides an overview of the effort necessary to be invested in SE education. He provides some perspective on the

status, but also discusses the historical development, and identifies some of the challenges faced by SE educators.

TABLE I. TOPICS (RELATED TO SE) FROM THE BAVARIAN COMPUTER SCIENCE CURRICULUM FOR THE 11TH GRADE [10].

10.3 Software Project	Combination of several modeling and implementation techniques (e.g. object-oriented programming and data base systems)
11.2.1 Software Engineering: concurrent projects	Project planning, software life cycle, process model, coordination of parallel processes
11.2.2 Software Engineering	Combination of different views, selected design patterns, analysis, design, implementation, test, documentation

B. Software Engineering at Schools

Software Engineering is more or less missing in schools, except in very few special secondary schools (e.g. the "Academy for Software Engineering" in New York [8]). In Informatics or Computer Science lessons, students typically learn to program based on worksheets or tasks in textbooks, but without getting to know even basics of the underlying software development process. In our opinion, it seems reasonable to embed teaching programming in the context of the whole SE process (and life cycle), which may, and this is our assumption, be useful for other subjects and cross-curricular school projects, too. But, only few secondary schools with a special orientation in computer science, business informatics or information technologies etc. follow curricula including SE or software development as a subject. 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]

And, what about the situation in Europe? Well, Bavarian secondary school students learn basics of software projects and SE in grades ten and eleven as part of the subject computer science [10]. It includes the software life cycle, different process models, planning, and covers the techniques from modeling through to the implementation in an object oriented programming language (see Table 1).

Not only in school curricula but also in relevant scientific literature, SE for secondary education seems almost non-existent, except some single initiatives like the master class for interested pupils at the Technische Universiteit Eindhoven (Netherlands). They organized a three days course in SE (six times) and concluded that the success of this master class shows that it fulfills a need [11].

In our opinion, too, there is a need of teaching software engineering already in secondary education because of different reasons. Firstly, there are subject-specific reasons. We believe that learning to program, which is part of numerous CS curricula, cannot be detached from SE. The students should learn at least basics of the whole software development process in order to get an overview. Already Aristoteles postulated in his 5th axiom that "the whole is more than the sum of its parts" [12]. This is supported from the field of neurodidactics, too. "People have enormous difficulty learning when either parts or wholes are neglected." [13] Hence, the learning brain needs the whole and the details; it "requires both a big picture and paying attention to the individual parts" [14]. The "big picture" in our case would be SE respectively the software development process, the individual parts would be the programming task.

Besides the subject-specific and neurodidactical reasons for introducing SE at schools, we argue that many aspects of this subject are part of general knowledge and/or are needed in several other fields and subjects. This is, what we have in mind when proposing SE contents for primary and secondary schools (see sections III B and IV).

C. Improving Software Practice through Education

According to the question of how to improve the field, Lethbridge et al. [15] 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 the following key challenges:

1) *making programs attractive to appeal good students and to meet societal demand,*
2) *understanding the dimensions of the field in order to focus education appropriately,*
3) *communicating real-world industrial practices more effectively to students,*
4) *defining curriculum standards that are forward-looking,*
5) *educating existing practitioners,*
6) *making SE education evidence-based,*
7) *educating educators, and*
8) *raising the quality and prestige of educational research.*

Lethbridge et al. analyzed the questions from the research aspects and formulated concrete research questions. Additionally, software education can be considerably improved by exemplary work on open source code topics. This is broadly treated in the literature, for example in the work of Rajlich's research group [16]. In addition, software process simulation modeling addresses a variety of issues related to the software development processes. Applications devoted to software process simulations cover processes narrowed to portions of the software life cycle but also cover larger evolutionary models. The summarizing work of Kellner [17] provides an overview of the most relevant works until 1999. It deals with the questions of "why" (purpose of the simulation), "what" (the scope and the variables that can be usefully simulated), and "how" (modeling approaches and techniques that can be efficiently utilized) simulation modeling is applied. The comprehensive summary of definitions is supplemented by guidelines and recommendations for selecting an appropriate simulation approach for practical situations. The paper of Zhang et al. [18] then provides comprehensive reflections of software simulation modeling until 2008. Their work also mentions the application of the SESAM tool that also served as basis for our own simulation environment, AMEISE (A Media Education Initiative for Software Engineering) which we use in Klagenfurt [19]. The work of Monsalve et al. [20] presents new

TABLE 2. THE SEEK KNOWLEDGE AREAS (KA) OF SOFTWARE ENGINEERING EDUCATION [25, P.21] (INCLUDING THE SUGGESTED AMOUNT OF TIME IN HOURS FOR TEACHING THE TOPICS).

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

trends in the application of game technology in SE education. Finally, Bollin and Samuelis [21] condense experiences gained during the experimentation with the AMEISE environment at the Technical University of Košice and its integration into the syllabi of its software engineering study program.

III. TEACHING SOFTWARE ENGINEERING

A. Software Engineering Knowledge Areas

There are various officially available syllabi that contain the SE topics we are referring to, including the Software Engineering Body of Knowledge (SWEBOK) [22]. This guide classifies SE into a set of 15 knowledge areas and it outlines the generally accepted topics in those areas. The available material was developed in a series of iterations over the past 15 years and passed through a rigorous series of review processes (with over 150 reviewers from 33 countries). The resulting document became an international standard through ISO/IEC JTC/SC7 [23], and it is now available in version 3.0.

For in-class education and for a course of instructions the collection is by far too abstract, but 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 [24]. As of 2014, there are five curriculum volumes covering the sub-disciplines "Computer Science", "Computer Engineering", "Information Systems", "Information Technology", and "Software Engineering".

The sub-discipline "Software Engineering" (also abbreviated as SE2004) [25] is commonly referred to by its chapter title "Software Engineering Education Knowledge" (SEEK). It is structured into Knowledge Areas (KA), which are broken down into units and topics. Table 2 summarizes these areas and provides a brief description (on a keyword level) of their content. Each knowledge area (in the detailed curriculum also each knowledge unit) is given a number of hours which correspond to the actual in-class time required. Knowledge units are not necessarily disjunctive. E.g., the entry "MGT" highlights the discipline of software project management, but several topics related to management (for example group dynamics) can also be found in other areas on a different level of abstraction and with varying theoretical background.

B. Software Engineering Syllabus at Austrian Schools

As already mentioned above, software engineering is almost non-existent in school curricula. This is the case in Austria, too, except for some secondary schools. These are vocational high schools of technology or business specialized in computer science and related fields.

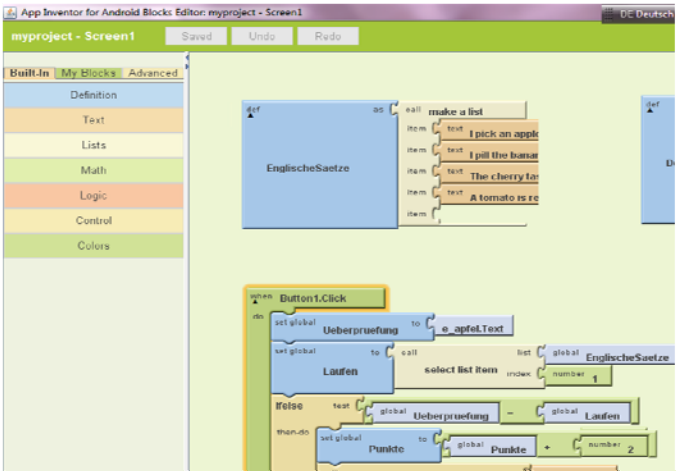
The curriculum of the "Höhere Technische Lehranstalt für Informatik" (Vocational High School for Informatics) contains the subject "Programming and Software Engineering" including, among others, the competence areas software design (DES), development, testing (PRO), and project management (MGT) [26].

The "Höhere Technische Lehranstalt für Informationstechnologie" (Vocational High School for Information Technology) teaches "Software Development" and "Information Technology Projects" including the competence areas project management (MGT), quality management (QUA), software development process and the implementation of IT-projects (PRO) [27].

The most detailed syllabus is given for the "Handelsakademie für Wirtschaftsinformatik" (Commercial Highschool for Business Informatics). The subject "Software Development and Project Management" includes the knowledge area software project management (MGT) and the following phases of the life-cycle (PRO) that are then to be covered:

- Project Initiation (Planning, Feasibility Study, Requirements Specification),
- Specification (Product specification by taking ergonomic requirements into consideration, requirements document),
- Design (Product design, Prototyping),
- Implementation (Modelling, Programming, Technical Documentation, Testing, Quality Assurance, Protocolling, System Generation),
- Delivery (Acceptance test including Protocols),
- Introductory Phase (Training, Manuals),
- Maintenance Phase (Helpdesk [28].

Fig. 1: App-Inventor (example screenshot of one project) – editor view, code of the "English App" (matching of German and English sentences) created by students [29]).



All other secondary schools do not teach SE, even if they have the topic "programming" in their curricula. Hence, it is reasonable integrating basic concepts of software engineering in form of projects. This is possible, too, as shown in the next section where we describe the project "App Programming with the App Inventor" [29], funded by the program "Teaching Informatics creatively", a regional part of the Austrian teacher support system IMST [1].

IV. The IMST Experiment

A. *Project Setting*

The aim of the project "App-Programming with the App-Inventor" was to create Smartphone Apps in one class of a high school, which were tested and evaluated by younger pupils from a lower secondary partner-school. The ideal opportunity for this was the Vocational High School of Commerce and Tourism in St. Veit, because some subjects that are being taught there are a good basis for Software Engineering projects: "Applied information technology", "Information and Office Management" and "Economics". The project was carried out in one class of grade 11 (which became our software engineers) in cooperation with a class of a lower secondary school in St. Veit (which became our clients, at grade 6). The project head chose these two classes because she is teacher in both, and wanted to strengthen the cooperation between the two schools. [29]

The vocational high school offers different branches of education, one for three years and two branches that lead to the qualification for university entrance (A-levels): Eco-Business (Environmental Studies) and International Management with three foreign languages (English, French, Italian), some classes with English as working language.

The students (totally about 500) are from 14 to 19 years old and can decide to learn up to six foreign languages. In addition to the obligatory languages they can choose Spanish, Russian and/or Slovenian. In addition, apart from that, the students get an extensive instruction in economics, "Information and Office Management" as well as in "Applied Informatics".

TABLE 3. EXAMPLE REQUIREMENT WORKED OUT IN THE CLASS (6TH GRADE), STEERING ALSO DISCUSSIONS ON GOOD AND BAD REQUIREMENTS [29]

Requirement	On Start of "English Exercise": Shows pictures and the user has to click on the correct word that describes the picture. Is it wrong, it changes to red, if it is correct, it becomes green and the uses sees a new screen with a new task

SE is not part of the curriculum and has never been taught before in that school. The idea to introduce it in one class has been deduced from the implementation of mobile learning. One informatics teacher then proposed to develop apps for Android devices, yielding a positive side effect: the chance to introduce all the steps necessary for software development and concepts prevalent in the field of SE. From a preliminary survey ([30]) we knew that also the technical equipment of the students was available. About 75 % of the pupils own a smart phone with touchscreen, 40% of them with the operating system Android. This allowed to work in pairs, which is reasonable, too, as the students got an introduction into pair programming, a commonly accepted programming technique in software engineering [31] as well as in education [32].

B. *Key Skills and Challenges*

The aims of the project where oriented towards the syllabi of the participating subjects: Applied Informatics with all related topics and competences on the one hand as well as the requirements of all subjects for which the apps were created. For the high school students the Applied Informatics topics of picture processing and databases where considered. The aim of the project was to create Smartphone Apps, mainly for learning and revising topics of other subjects. During this project they went through a shortened software engineering process resulting in a mobile application they could use on their own smartphones and evaluated them together with their young colleagues [29].

As none of the students had any programming knowledge, the teachers decided to apply the App-Inventor from the MIT (Massachusetts Institute of Technology) [33]. With this software, Apps for Android-devices can be created easily – and without programming experiences. Like in other educational software systems (for example Scratch [34]), the code can be built out of predefined parts including some concepts of programming (see Figure 1). Nevertheless, it was a challenging situation for both, the students and the teachers, because of missing previous knowledge as well as some organizational problem. The students had to work a lot on their own and teachers tried to help and to overcome difficulties.

In the preparation phase of the project, the students got an introduction to the topics relevant for app programming and prescribed in the informatics curriculum: databases, picture editing and processing. Like for every new tool, also for the App-Inventor, an introduction for the students was necessary. This was accomplished by a short teacher presentation as well as some exercises and adequate solutions.

Fig. 2: Test and Evaluation of the App (according to a pre-defined questionnaire)

TABLE 4. THE SEEK KNOWLEDGE AREAS OF SOFTWARE ENGINEERING EDUCATION [25, P.21] (THE AREAS MOSTLY COVERED BY OUR IMST PROJECT ARE SHADED IN Gray).

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

In this way, the students were able to get a first view of these concepts and found out how programming can work.

After this short introduction phase, the main part of the project started during which the students had to fulfill the following steps:

(1) planning the working process,

(2) designing an own App,

(3) collecting or creating materials,

(4) using the App-Inventor to realize the Apps, and

(5) testing and evaluating the Apps.

During one lesson, both cooperating classes from the vocational high school and from the lower secondary school met to get known each other. The main objective of this lesson was requirements engineering. The high school students discussed possible topics and requirements of the apps that should be developed. This was the start of step (1) *planning the working progress.* The students wrote down their ideas and the requirements of the apps they wanted to be developed (see Table 3 for a small example – yielding also discussions on the granularity level and quality of requirements).

Based on this list, the students designed and modeled their App (2) and collected and/or created (3) materials necessary for the App (e.g. a database of Spanish vocabulary). Then, they programmed their apps in the App Inventor (4), and finally tested and adapted them. At the end, all Apps worked well and were presented in the lower secondary class where all students were excited about them. High school and secondary school students tested and evaluated them together, based on pre-defined questions (5) (see Figure 2).

Referring to the SEEK knowledge areas of software engineering education [25, p.21] fundamentals of the following key areas were covered in the project (see Table 4 for a summary):

- The school students (6th and 11th grade) heard about the *idea of a software development process* and the influence on the final product. They learned about *how to start* and which *steps* are leading to a working version of the App (PRO).
- The students from the 11th grade got some training (repetition) on *software fundamentals*, and the classical notions of *blocks, logic and control* were repeated before their project started (FND). They also got an introduction to the task setting and the development environment (CMP).
- They (6th and 11th grade) spent a lot of time in *planning their project* as they had to *synchronize* with another class (client resp. company) and had to agree about the *schedule*. They also had to decide about *when something* has to be done by *whom* (MGT).
- The students (6th and 11th grade) trained their *communication skills* in written and oral form. They learned how to ask for requirements and how to *interact* with the "client" and the "engineers" (PRF).

Fig. 4: App "Magic Ball", screenshot of THE Android device, with a field called "Frag den Magic Ball …" ("Ask the Magic Ball" [29]).

- The students from the 11th grade had to write down the requirements in form of a specification (English text) in a concise manner and discuss them with the students from the 6th grade (MAA).
- They (11th grade) spent a lot of time in designing the human computer interface, thought about different interaction strategies, and thought about how the system should look like on the system design level (DES).
- The students (6th and 11th grade) had to think about how to test for and ensure the quality of the developed App. Apart from reviews and testing activities to be planned, they worked out a questionnaire to get an answer to this question. It was also defined how to deal with problems that arise during the project (VAV and QUA)

At the end of the project, each group had a running smartphone App (for an example, see figure 4) and all of them were presented and evaluated in the "evaluation class". The vocational school students got a very positive feedback, which proved that it was worth the effort and they could be proud of their own self-made products.

For the younger pupils it was very exciting to test the Apps and they were fascinated that the older students could produce such Apps on their own.

C. *Evaluation & Results*

The evaluation of the project is inspired by the "Angebots-Nutzungs-Modell" (process-product paradigm for the evaluation of teaching quality) [35]. This model considers different aspects of teaching in the evaluation, from *starting conditions* (demographic variables, cognitive preconditions etc.) over *process variables* (learning progress etc.) to *outcomes* (learning outcomes, completed apps).

TABLE 5: THE TARGET GROUPS OF THE IMST EXPERIMENT AT THE VOCATIONAL HIGH SCHOOL OF COMMERCE AND TOURISM IN ST. VEIT/AUSTRIA AND THE LOWER SECONDARY SCHOOL IN ST. VEIT/AUSTRIA

School	Grade	Role	Girls	Boys	total
High	11	software engineers	8		8
Secondary	6	clients	10	15	25

With regard to the project aim of introducing SE fundamentals in the vocational high school and due to the small group (eight girls, see Table 5 for the details) we focused on a qualitative evaluation. We were mainly interested in the qualitative feedback of the "software engineers" respectively the "programmers" in order to check the acceptance and feasibility.

The methods we used were observation of the processes during the lessons (programming and communication), open interviews as well as a self-constructed questionnaire (evaluation of the project and the products).

The results of the observation from teachers and students can be divided as follows:

(1) Student-related aspects:
The high school students were highly motivated and interested in programming their own apps, although it is not part of the syllabus. Only when the project was first proposed to the students, some of them said, that programming would be too difficult for them. This corresponds to the different empirical findings concerning the low self-concept of girls in technical fields [36]. But, the first fears could be easily dispelled and the students were surprised about their own performance and the good evaluation of their apps, which had an important effect (only observed, not measured) on their self-concept towards informatics. The communication skills of the students were different, in part due to age and gender, but also group size. The students of the lower secondary school often were not concentrated and loud, but the communication was getting better and better with the duration of the project.

(2) Subject- and process-specific aspects:
Most of the students worked very independently and did not need the help of the teacher. Even those, who sometimes asked for help were able to finish the apps on their own. The students were aware of the importance of defining the requirements and spent much time on this point. These requirements were formulated in everyday language and, firstly, the students wanted too much. So the teacher had to help in order to reduce and formulate the requirements. Some students then had difficulties in finding the right algorithm for their app, which was expected, as they had never programmed before. ("Developing the app was not difficult, I only had problems in defining the different steps and the correct order.") During the programming phase they

mainly had difficulties in the handling of App- Inventor ("The buttons didn't appear as I wanted them.").

(3) Product-specific aspects:
Some students of the 6th grade found that some apps should have included more tasks, but generally they liked all apps and praised them. One student even mentioned: "Hat off and respect for the great performance of the programmers."

The most frequently mentioned answers to the three open questions are presented here (translated or summarized by the authors):

1.) What did you like most in App programming?
- I liked to try out all the Apps.
- It was great that we could be creative and it was fun to fiddle about with the App Inventor. Further it was interesting to take a look at other Apps and to compare them with our own. It was interesting to see how much work it was to create our small Apps.
- I liked to choose the pictures and sentences by myself, so I could be very creative.
- The work with the App Inventor was difficult but I liked it very much to plan and create Apps self-reliantly.

2.) What didn't you like in app programming? They didn't like
- that some errors could not be fixed.
- that it took much time to adapt the apps and
- it was exhausting and a bit annoying to type in all correct as well as all wrong solutions.
- to put together all the needed parts in the App Inventor.

3.) Did you have problems in programming?
- It was not possible to close the Apps.
- It was easy to generate the Apps but it was complicated to put together all the steps in the Block-Editor because we had to fade out all the English answer-possibilities.
- The buttons did not look like I wanted because they were on the wrong place. I was not able to make pictures appear and disappear at the right time. It was not possible to create a break button and nobody could help me with this problem. Nothing worked as it should.
- It was difficult to program the buttons with the correct form and functioning.
- Our teacher could help me with every difficulty and all problems could be solved.

V. DISCUSSION

The observation of the teachers showed that despite the missing programming competences and knowledge about the concepts of SE the students created good and useful apps that fulfilled the requirements they listed in the planning phase. There was a sort of "intuitive software engineering" that one of the authors (also Spanish teacher in the high school class) could observe. When the students were asked before the project, how they would proceed in the development of their app, they defined the following steps:

- We must first know what the app shall do (requirements).
- Then we design the screen (modeling) and
- plan who will do what and when (project management).
- We program the app (programming) and
- try and change it until it works (testing and debugging).
- Finally we "sell it" to you and show you how to use it (implementation).

After this last statement they asked the Spanish teacher, what the app should be able to do and which contents they should integrate (vocabulary or grammar etc.). We discussed in intensively, which shows, that they were aware of the importance of "specification" without knowing the concept.

Reflecting all qualitative data from our interviews and observations we postulate that fundamentals of software engineering can definitely be introduced in secondary education without difficulties.

But, what is the ideal time for the start? As every subject has some fundamental ideas [37, 38] that can be taught at every age (certainly in an appropriate way and extent) it seems necessary to identify them in all knowledge areas of Software Engineering and to divide them in subject-specific aspects like programming and general aspects, that are relevant for (and could be linked to) other subjects, too, like *communication skills*, *logic*, *modeling* or *problem analysis*. It has to be specified which aspects and knowledge areas can be taught in which subject and grade. Then, we could start already in primary education, as the preliminary data of our research project "Informatics – A Child's Play" shows. We could successfully teach e.g. fundamentals of logic (truth table, AND, OR, NOT) already in pre-school and modeling (entity relationship diagrams, class diagrams and flow charts) in the grades 2-3 [39].

VI. CONCLUSION

This paper demonstrates 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) are presented, and related to the tasks the students (6th and 11th grade) had to fulfil. However, the approach applies to other grades as well, and in the future we will draw our attention to other school levels, too.

Due to the small number of participants, we present the results from the qualitative feedback on anecdotal basis only, but, in our setting it turned out that quite a lot of the topics related to the software engineering knowledge areas were touched by the experiment. The results might have some bias (curiosity of younger pupils, different teachers …), but, with this first quite encouraging experiment, we hope to approach a new field of research, and to initiate further discussion.

Finally, as the project is still ongoing – including also lectures at primary schools – we hope that with the lessons learned from this first experiment, we are able to teach Software Engineering concepts already in primary schools, and thus help in increasing the enthusiasm for technical studies again.

REFERENCES

[1] IMST Project Homepage, Waltraud Rohrer, 2014. [Online] https://www.imst.ac.at/texte/index/bereich_id:8/seite_id:8. [Accessed: Nov. 12th, 2014].

[2] B. Meyer, "Software Engineering in the Academy", IEEE Computer, pp. 28-35, May 2001.

[3] B. Meyer, "Reality: A cousin twice removed", IEEE Computer, pp. 96-97, July 1996.

[4] B. Meyer, "Testable, Reusable Units of Cognition", IEEE Computer, vol. 39, no. 4, pp. 20-24, April 2006.

[5] A. Pyster, R. Turner, D. Henry, K. Lasfer and L. Bernstein, "Master's Degrees in Software Engineering: An

[6] P. N. Robillard, "Opportunistic ProblemSolving in Software Engineering", IEEE Software, pp. 60-67,

[7] G. W. Hislop, Software Engineering Education: Past, Present, and Future, IGI Global, 2009.

[8] AFSE Homepage, The Academy For Software Engineering, 2014. [Online] http://www.afsenyc.org. [Accessed: Nov. 12th, 2014].

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

[10] Hubwieser, P. Computer Science Education in Secondary Schools--The Introduction of a New Compulsory Subject. ACM Transactions on Computing Education (TOCE), 12(4), 16. 2012.

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

[12] Aristotle, Hermann Bonitz. Aristotelis Metaphysica. BiblioBazaar. 302pp. 2009.

[13] Caine, R. N., & Caine, G. 12 Brain/Mind Natural Learning Principles (1990). [Online] http://www.cainelearning.com/wp-content/uploads/2014/04/12-Brainmind-principles-expanded.pdf (Accessed: Nov. 19th, 2014)

[14] Caine, R. N., & Caine, G. (1990). Understanding a brain-based approach to learning and teaching. Educational Leadership, 48(2), 66–70.

[15] 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, 23-25 May 2007.

[16] J. Buchta, M. Petrenko, D. Poshyvanyk and V. Rajlich, "Teaching Evolution of Open-Source Projects in Software Engineering Courses", in Proceedings of the 22nd IEEE International Conference on Software Maintenance, pp. 136-144, 2006.

[17] M. I. Kellner, R. J. Madachy and D. M. Raffo, "Software Process Simulation Modeling: Why? What? How?", Journal of Systems and Software, vol. 46, no. 2/3, pp. 1-18, 15 April 1999.

[18] H. Zhang, B. Kitchenham and D. Pfahl, "Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review", in Making Globally Distributed Software Development a Success Story, Springer Berlin/Heidelberg, Lecture Notes in Computer Science, Q. Wang, D. Pfahl and D. Raffo, Eds., 2008, pp. 345-356.

[19] R. T. Mittermeir, E. Hochmüller, A. Bollin, S. Jäger and M. Nusser, "AMEISE – A Media Education Initiative for Software Engineering: Concepts, the Environment and Initial Experiences", in Proceedings International Workshop ICL – Interactive Computer Aided Learning, Villach, Sept. 2003.

[20] E. S. Monsalve, V. M. B. Werneck and J. C. Sampaio do Prado Leite, "Teaching Software Engineering with SimulES-W", in Software Engineering Education and Training (CSEE&T), Honolulu, HI, 22-24 May 2011.

[21] L. Samuelis and A. Bollin, "Experiences gained in teaching software project management by simulation," in Informatics 2009: Proceedings of the Tenth International Conference on Informatics, Herľany, Slovakia, November 23-25, 2009.

[22] P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014. [Online] www.swebok.org. [Accessed: Nov. 10th, 2014].

[23] IEEE, "Guide to the Software Engineering Body of Knowledge - SWEBOK," Technical Report, ISO/IEC 19759:2005, Software Engineering, 2005.

[24] IEE/ACM Joint Task Force on Computing, "Computing Curricula 2005, the Overview", 2005. [Online] http://www.acm.org/education/curricula-recommendations. [Accessed: Nov. 10th, 2014].

[25] ACM IEEE, "Software Engineering 2004", 23 August 2004. [Online] http://sites.computer.org/ccse. [Accessed: Nov. 11th, 2014].

[26] Lehrplan der Höheren Lehranstalt für Informatik. [Online] http://www.htl.at/fileadmin/content/Lehrplan/HTL_VO_2011/BGBl_II_Nr_300_2011_Anlage_1_4.pdf

[27] Lehrplan der Höheren Lehranstalt für Informationstechnologie. [Online] http://www.htl.at/fileadmin/content/Lehrplan/HTL_VO_2011/BGBl_II_Nr_300_2011_Anlage_1_5.pdf [Accessed: Nov. 19th, 2014]

[28] Lehrplan Handelsakademie für Wirtschaftsinformatik – Digital Business.[Online] https://www.hak.cc/files/syllabus/LP_Digbiz_131219.pdf [Accessed: Nov. 19th, 2014]

[29] Stadtmann, C. App-Programmierung. Unpublished project report. Alpen-Adria Universität Klagenfurt. 2013.

[30] B. Sabitzer and E. Bischof, „Mobile Language Learning. In: New Perspectives in Science Education. Proceedings of the Future of Education Conference, Simonelli Editore - University Press, Florence, Italy 8-9 March 2012.

[31] Agile Software Development with Scrum. K. Schwaber, M. Beedle. Prentice Hall, 2001.

[32] S. Faja, "Pair programming as a team based learning activity: a review of research." Issues in Information Systems 12.2: 207-216. 2011.

[33] App-Inventor. [Online] http://appinventor.mit.edu/. [Accessed: Nov. 19th, 2014]

[34] Scratch Homepage. [Online] http://scratch.mit.edu/ [Accessed: Nov. 19th, 2014]

[35] Helmke, A. & Schrader, F.-W. Vom Angebots-Nutzungs-Modell zur Unterrichtsentwicklung. In A. Bartz, H.-J. Brandes & S. Engelke (Hrsg.),Praxishilfen für die mittlere Führungsebene in der Schule. Modul 3: Unterrichtsentwicklung (S. 3-6). Köln: Carl Link Verlag. 2011.

[36] R. Buhr. Genderkompetenz in technischer Bildung, für „Gatekeepers“ und im öffentlichen Raum. In: Buhr (ed.), Technische Bildung für alle. Ein vernachlässigtes Schlüsselelement der Innovationspolitik. Bundesministerium für Bildung und Forschung (BMBF), Berlin. 2008.

[37] Bruner, J.S. Toward a theory of instruction. Harvard University Press, Cambridge MA. 1966.

[38] Schwill, A. Ab wann kann man mit Kindern Informatik machen? INFOS2001-9. GI-Fachtagung Informatik Und Schule GI-Edition, pp.13–30. 2001.

[39] Sabitzer, B.; Antonitsch, P.; Pasterk, S. (in press). Informatics Concepts For Primary Education: Preparing Children For Computational Thinking. Submitted to WiPSCE 2014 - The 9th Workshop in Primary and Secondary Computing Education. November 5-7, Berlin, Germany. 2014.