Teaching Software Engineering Principles in Programming and Non-Programming Activities at Schools

Claus Pahl
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy

Ilenia Fronza
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy

***Abstract*—Software engineering principles are relevant to students at school, irrespective of whether they will become professional developers. We discuss two types of situations for teaching software engineering principles that we covered in a number of publications. Firstly, we discuss app development and robotics examples as more traditional settings for software development. Secondly, we show how the creation of infographics and videos represents an opportunity to promote agile approaches. In schools, computer science subjects are often taught in the context of other subjects, which we address through the second situation above. We focus on a range of activities that aim at fostering software engineering principles, while also considering the achievement of the existing curricular learning objectives.**

***Index Terms*—Software Engineering; Software Engineering Training and Education; End-User Software Engineering.**

I. Introduction

Bollin et al. [6] looked at the need for teaching Software Engineering (SE) principles in schools. Their feasibility analysis revealed that this is effective in training skills needed today beyond the actual software development. These skills include for instance group dynamics, psychology and communication skills as well as general soft skills, but also logic, planning, modelling, and computational thinking as a problem solving ability. End-User Software Engineering (EUSE) [7] refers to non-professional, untrained developers – thus a definition that applies to students at school from primary to high school levels. The challenge of EUSE in schools is often understanding how to leverage existing curricular activities (that do not have software development as their main objective) to foster Software Engineering principles if focussed programming courses are not part of the curriculum.

Central in this effort for schools are the acquisition of software quality notions from a product perspective, but also SE principles from a process perspective. This aim creates a number of challenges. Often, direct SE lectures should be avoided as school pupils often do not realise the usefulness of learning SE principles. Furthermore, when computing topics are taught as part of other subjects, meeting their core curricular learning objectives becomes difficult.

The work presented here has been supported by the COCONATS project (https://coconats.inf.unibz.it/), Free University of Bozen-Bolzano.

We consider teaching formats that aim at achieving computer science objectives and specifically foster SE principles, while also guaranteeing the achievement of the existing curricular learning objectives. Here, EUSE can aid the student's goals, without aiming to transform them into professional software engineers with targeted direct courses.

The EUSE field has largely focused on adult, mainly professional contexts, only a few contributions are dedicated to schools. Most of these studies target the teaching of agile methods, which turn out to be suitable for schools. For example, we propose a framework in which a series of agile methods can be adapted for middle schools. We have published on the two settings outlined [1]–[4] and will describe the main approach and insights here. We will cover programming and non-programming activities.

II. Software Engineering Principles in Schools

In order to bring the benefits of a Software Engineering (SE) approach to end-users, Burnett and Myers recommend to consider and respect the student's (i.e., end-users in EUSE terminology) goals and working style [8]. The latter can often be opportunistic and incremental, collaborative, and organised into trial-and-error phases. Agile methods are consequently a good candidate in this endeavor, simply because they inherently favor a flexible, iterative approach. Their focus is also more on the product than on the production of documentation. The latter would negatively impact on students' motivation.

The existing attempts to introduce agile principles to schools have been compared with a long waterfall development approach. Though here the focus was more in the final product, rather than the process through which students developed the product. Adopting agile, as we propose, fills this gap. This is also more transferable, e.g., to non-STEM subjects, since the agile focus is less in technical product quality. As a consequence, software engineering principles also become relevant for non-technology oriented students. Software engineers do not just write programs; they think in terms of satisfying needs and solving problems. Moreover, following a (software) engineering process teaches how to manage all the steps from initial customer inception to the release of the finished product.

The selection of infographics and videos for our sample courses had the following reasons. 1) They are transversal to many disciplines, and can be adapted to different types of schools, e.g., creating videos and infographics takes into account the need for visual communication skills. 2) They also engage them to learn subjects they are not particularly gifted at. Videos and infographics are good candidates, as the current generation of students has grown up with digital media.

III. Approach

In this work, we focus on middle and high schools, covering an age range of 10 to 17 years, with the additional goal of connecting to students before they choose their future career. However, apart from enticing students into computer science studies, we realise that not all will become professional developers, but will nonetheless need to be equipped with capabilities suitable for a digitalised world.

We discuss here two different types of activities:

- apps and robots as computing topics, with a clear link to programming and development [5],
- information and media, as concerns of the wider digital world, without any concrete software connection [3], [4].

A. Robotics, Apps and Blogs

The goal of one proposed teaching module is completing a robotics project. This is accompanied by documenting these development activities in a blog, which is often also of curricular value in school. Writing a blog can link to the students' interests and social communication habits, and thus engage them better in the technical work. Here, using smart phones or other mobile devices to document project activities also helps towards a more competent and responsible use of these devices. The blogs are anyway a good opportunity for an integration in a traditional curricula.

A module focusses on combining robotics with digital media, consisting of four main parts:

Introduction and team creation (4 hours);
Blogs: introduction and first posts (6 hours);
Unplugged introduction to robotics (4 hours);
Robotics project (6 hours).

Here, from a software engineering perspective, a software process is clearly applicable. This could be in the form of agile methods with a focus on early implementation and testing, with intensive interaction with the 'customer', which is the instructor in this case. The challenge is here the assessment. Generally software quality aspects could be applied, more research is here needed to develop assessment methods that will work in a school's curriculum.

B. Digital Media

We applied a set of agile (eXtreme Programming XP) practices in these courses: a) Incremental development, through small releases, frequent testing, and user stories; b) Customer involvement, by having the instructor playing this role; and c) Change, through regular system releases, test-first, refactoring, and continuous integration.

The infographics and video course is organised as follows:

Introduction (2h): tips on infographics/videos but no SE/Agile
Paper-based prototype (6h): small releases (20-min cycles), user stories (storyboards), on-site customer (10-min with teacher)
Creation of infographic (4h): small releases (20-min cycles), testing, on-site customer (10-min revision) + peer evaluation
Presentation (2h): conformance with the initial requirements and paper-based prototype user stories and testing

Based on frequent small releases, it provides an opportunity for formative assessment. Focusing on the process side (rather than on the product), we have selected a set of XP practices, which are recommended to be adopted together. The table below shows the XP practices that can be implemented.

Practice	Description
On-site customer	The customer (i.e., teacher) is always present to provide continuous and direct feedback.
Testing	Acceptance testing (to allow continuous feedback), and test-first (to test as soon as possible thus minimizing long-term testing costs).
User stories	Informal prototypes (e.g., storyboards) that describe requirements.
Small releases	Decomposition of activities in short iterations in order to obtain timely and continuous feedback.

IV. Results and Conclusions

We introduced two separate contexts in order to foster software engineering skills and principles in schools: a computing-oriented robotics/apps module and a digital media-oriented creation of infographics and videos. In both cases, this was an opportunity to promote process-oriented principles (here agile) in middle and high schools.

Our assessment included both product and process aspects. We observed a quality improvement of the product throughout the iterations. The results also show that students started adopting a software engineering approach, and managed to organize their activities in order to be able to present a prototype at the end of each iteration.

Our future work includes the further verification of how fostering SE practices through programming and non-programming activities can be beneficial to students when they start programming. More experiments would be beneficial to further clarify the effectiveness.

References

[1] I Fronza, N El Ioini, C Pahl, L Corral. Bringing the Benefits of Agile Techniques Inside the Classroom: A Practical Guide. Agile and Lean Concepts for Teaching and Learning, 133-152. 2019.
[2] A Colombi, I Fronza, C Pahl, D Basso. COCONATS: Combining Computational Thinking Didactics and Software Engineering in K-12. 19th SIG Conference on Information Technology Education. 2018.
[3] I Fronza, C Pahl. End-User Software Engineering in K-12 by Leveraging Existing Curricular Activities. ICSOFT, 283-289. 2019.
[4] I Fronza, C Pahl. Teaching Software Engineering Principles in Non-Vocational Schools. CSEDU. 2019.
[5] I Fronza, L Corral, C Pahl. Combining Block-Based Programming and Hardware Prototyping to Foster Computational Thinking. SIG Conference on Information Technology Education (SIGITE '19). 2019.
[6] A. Bollin, S. Pasterk, P. Antonitsch, B. Sabitzer. Software engineering in primary and secondary schools-informatics education is more than programming. Intl Conf on SE Education and Training, 2016.
[7] S. Chimalakonda, K. V. Nori. What makes it hard to teach software engineering to end users? some directions from adaptive and personalized learning. IEEE Conf on SE Education and Training, 2013.
[8] M. M. Burnett, B. A. Myers, B. A. Future of end-user software engineering: beyond the silos. In Future of Software Engineering. 2014.