End-User Software Engineering in K-12 by Leveraging Existing Curricular Activities

Ilenia Fronza and Claus Pahl
Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100, Bolzano, Italy

Keywords: End-User Software Engineering, K-12, High School.

Abstract: In recent years, an increasing number of people (called "end-users") have started to perform a range of activities related to software development, such as coding with domain-specific languages. The research in the area of End-User Software Engineering (EUSE) aims at improving the quality of end-user-produced software by paying attention to the entire software life cycle. The increasing number of activities dedicated to the diffusion of coding in K-12 motivates the need of EUSE also in this environment. Students indeed will probably need to produce software in their future careers (even if not professionally), and the quality of their software may be crucial. In this work, we describe a didactic module in which the activities usually carried out in the existing study programme are exploited to introduce software engineering principles. The module does not shift the students' attention from their main objectives and does not introduce additional lectures on software engineering topics. We describe the results of a first edition of the module that involved 17 students in a trilingual international high school. The results are promising and allow us to formulate hypotheses for further work, such as extending our approach to other activities and observe if and when students will develop a "software engineering mindset", even without developing software.

1 INTRODUCTION

The gap between end-users and professional software developers has narrowed noticeably: an increasing number of people not only use software, but also participate in the development process in order to solve different types of problems. The evolution of the meaning of the term *end-user* well represents this phenomenon. Indeed, the term end-user has been initially introduced as antonym of "professional developer", in order to distinguish those who *use* software systems in their daily activities from those who *create* software systems professionally. Today, the term end-user encompasses a wide range of software-related activities (Burnett and Myers, 2014), such as software development with domain-specific languages (Ye and Fischer, 2007).

The disadvantage of this phenomenon is the overall low quality of end-user-produced software. The research field of End-User Software Engineering (EUSE) addresses this issue by looking beyond the "creation" part of software, and paying attention to the rest of the software life cycle (Burnett, 2009).

In the last decade, computational thinking has been recognized as part of the key skills that must be acquired by all students, regardless of the degree and course of study (Wing, 2006). For this reason, activities are increasingly often proposed in K-12 to strengthen computational thinking skills, in all disciplines and also in non-vocational schools, where students are less inclined to Science, Technology, Engineering and Mathematics (STEM). The aim is to prepare students for the characteristics of the current labour market: information technology plays an important role, independent of the chosen career, therefore being able to creatively use technology to solve problems (rather than being passive users) could make a difference. If these activities in K-12 will achieve their objectives, we can expect more and more end-users to be engaged in non-professional programming activities in the future (Fronza et al., 2014).

For this reason, it is important to equip students with the necessary means to improve software quality: in their future career, developing (although not professionally) reliable software could be crucial. This means that EUSE, which aims at bringing the benefits of a Software Engineering (SE) approach to end-users, finds an application also in K-12 to improve the quality of students' software. Moreover, students can benefit from the application of SE prin-

Fronza, I. and Pahl, C.
End-User Software Engineering in K-12 by Leveraging Existing Curricular Activities.
DOI: 10.5220/0006846702490255
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 249-255
ISBN: 978-989-758-320-9

ciples in other fields: for example, the ability to work in an iterative and incremental way can be considered transversal to many disciplines, and also to everyday life.

Nevertheless, bringing SE principles to end-users (and therefore also in K-12) sets a number of challenges to EUSE's researchers (Chimalakonda and Nori, 2013). For example, end-users often do not understand the usefulness of learning SE principles, and are focused exclusively on their specific objective (i.e., solving a specific problem). In this regard, the EUSE approach is to respect end-users goals and working habits, without aiming to transform them into professional software engineers with ad-hoc courses (Burnett and Myers, 2014).

In this work, we describe a didactic module in which we leverage the activities usually carried out in the existing study programme to bring SE principles inside the classroom. The module does not shift students' attention from their main objectives and does not introduce additional lectures on software engineering. This aspect is of paramount importance in non-vocational schools. In these schools, infact, on one hand one wants to create a mindset that could improve the quality of any software produced in the future; on the other hand, additional lectures on SE would not be perceived as "useful", especially by those students who are less inclined to STEM.

We describe the results of the first edition of the module that involved 17 students in a trilingual international high school in the city of Bolzano (Italy). The results are promising and allow us to formulate hypotheses for further work, such as extending our approach to other activities and observing if and when students will develop a "software engineering mindset", even without developing software.

Section 2 describes the state of the art of EUSE in primary and secondary schools; Section 3 describes the rationale of the proposed didactic module, and Section 4 details its structure. Section 5 describes the first edition of the module, and Section 6 shows its results. Section 7 discusses our results and draws conclusions from this work, also proposing possible directions for future work.

2 EUSE IN K-12: STATE OF THE ART

The field of End-User Software Engineering is quite recent (Burnett, 2009), and only a few existing studies in the field are dedicated to the specific case of primary and secondary schools.

Meerbaum and Hazzan presented a mentoring methodology on Agile for high schools (Meerbaum-Salant and Hazzan, 2010).

In 2015, Fronza et al. designed and implemented a course, in which the phases of the software development process are leveraged to promote computational thinking learning (Fronza et al., 2015).

The work of Bollin et al. underlined the need and feasibility of teaching SE principles in K-12. According to the authors, SE can be a valuable means to exercise a set of skills that are needed nowadays. These capabilities include: group dynamics, psychology, communication skills, logic, planning, modelling, and computational thinking as an ability to solve problems (Bollin et al., 2016).

In 2016, Kastl et al. achieved greater flexibility in software development projects in three secondary schools in Germany, by applying Agile methods in class (Kastl et al., 2016). Fronza et al., in 2017, focused on teaching SE to end-users, by proposing a framework in which a series of Agile practices have been adapted to the context of middle schools to teach computational thinking (Fronza et al., 2017).

Teixeira Monteiro et al. have analyzed how the technology used in their programme can introduce SE elements in the software created by the participants (Monteiro et al., 2016).

In this work we describe a didactic module that we have designed to bring Software Engineering principles inside the classroom in the first year of a non-vocational high school. In order to respect end-users goals and working habits, the module fosters software engineering concepts by focusing only on the process side and we do not introduce additional lectures on software engineering.

3 RATIONALE OF THE DIDACTIC MODULE

One of the challenges in the End-User Software Engineering (EUSE) research field is to identify the most suitable software development process for each specific type of end-user (Chimalakonda and Nori, 2013). In this regard, Burnett and Myers recommend to respect end-users' goals and working style, which is preferably opportunistic and incremental (Burnett and Myers, 2014), collaborative (Costabile et al., 2008), and by trial-and-error phases (Burnett and Myers, 2014).

This description brings back to mind the philosophy of Agile software development, which favors a flexible, iterative approach, and focuses more on the product than on the production of unnecessary documentation.

Recently, the idea of introducing Agile in the educational context has attracted researchers' and educators' attention, and some works have been produced to report some experiences in this direction (see Section 2). However, these works confront an environment (school and university) in which the waterfall development model has been for a long time the most adopted (and taught) development strategy (Kropp and Meier, 2013; Kastl et al., 2016). Adopting an Agile approach, in fact, would require a considerable effort to switch to an environment in which the process by which students arrive at the product is taken into consideration even more than the final product itself (Steghöfer et al., 2016).

This shift would provide, among its advantages, the chance of introducing SE principles even in schools where students are less gifted at STEM subjects. This would allow us to reach a larger number of students, which is in turn important: in the current labour market, indeed, there is a high probability to face the need of creating software in a number of different careers (Ye and Fischer, 2007).

The challenge is understanding how to leverage existing curricular activities (that do not have software development as a main objective) to foster SE, without introducing specific lessons on the topic, thus respecting the students' objectives.

In this work we describe a didactic module that we have designed to foster software engineering principles in the first year of a non-vocational high school. The proposed module covers six hours of curricular activity, and leverages activities that are usually carried out in the existing study programme (and do not include software development). Therefore, following the EUSE guidelines (Burnett and Myers, 2014), our module does not shift the students' attention from their main objective and does not introduce additional lectures on software engineering.

3.1 Methodologies and Practices

Extreme Programming (XP) is an Agile methodology that integrates practices related both to project management and to development process, by focusing on continuous communication and programming practices (Beck, 2000). Our didactic module adopts XP as a methodology, for two main reasons:

- it doesn't require end-users to radically change their work habits (see the introductive part of this section);
- it helps end-users to organize their process by introducing a series of light practices.

The positive aspect of XP is that it provides a set of principles and practices to guide the development process. Depending on the specific context and objectives, only a limited set of practices might be selected. However, it is important to take into account the existence of hidden dependencies among these practices. For example, the application of collective ownership of the code in isolation can lead to a chaotic situation (Fronza et al., 2018), which can be mitigated by adopting other practices, such as pair programming.

In the specific case of our didactic module, which focuses on the process aspect, it is suggested to apply together the following XP practices (Fronza et al., 2018):

- *On-site Customer*: the customer is always present during the development process to provide continuous and direct feedback.
- *Testing*: different types of testing strategies are possible, such as acceptance testing (to allow continuous customer's feedback), and test-first (to test as soon as possible and thus minimize the cost of long-term testing).
- *User Stories*: the objective of these informal prototypes is to describe requirements in a language that is understood both by the team and the customers.
- *Small Releases*: decomposition of software development activities in short iterations in order to obtain timely and continuous feedback.

The next section illustrates how we have structured our didactic module in order to promote these practices while performing one of the activities of the existing syllabus, namely the creation of a set of slides for a presentation. It should be noted that, for this purpose, we do not move the students' attention away from their main objective, and we do not provide additional lessons on SE.

4 STRUCTURE OF THE DIDACTIC MODULE

The task of creating a set of slides for a presentation (usually at the end of a project or activity) is often considered a pure exercise of computer literacy; at the end, the teacher evaluates the quality of the presented slides (i.e., the product), but the process needed to create these slides is almost ignored. Being a very common task, and trasversal to many disciplines, we considered this task as a good candidate to be the focus of our didactic module.

The initial requirement needs to be deliberately vague (e.g., "prepare a presentation about the topic *X*"), so as to require students (as in the case of software engineering) to make an effort to understand the problem and formulate a solution.

In total, this activity covers *six hours* (for example, four blocks of 90 minutes). The remaining part of this section details the structure of the module and the assessment strategy.

4.1 First Part

The first part includes the following activities:

1. Creation of a mind map (paper based exercise) to better organize ideas within the topic of the presentation (for example: what sub-topics can be discussed in the presentation?), and to identify their relevance, strength, and impact, as well as to describe the relationships between ideas. This activity requires 20 minutes, and the relevant XP practice is *user stories*.
2. Revision of mind maps, together with teachers. This activity requires 15 minutes, and fosters the following XP practices: *on-site customer, small releases*.
3. Searching the necessary information (e.g., online) in order to expand the topics in the mind map. This activity requires 20 minutes, and the relevant XP practice is *small releases*.

The second and the third step are repeated twice (i.e., two iterations are performed).

4.2 Second Part

The second part includes the following activities:

1. Preparation of a set of slides about at least two of the topics in the mind map. This activity requires 15 minutes, and the relevant XP practice is *small releases*.
2. Teachers' feedback. This activity requires 10 minutes, and the relevant XP practices are *on-site customer* and *testing*.
3. Continuing the preparation of the set of slides. This activity requires 20 minutes, and the relevant XP practice is *small releases*.
4. Teachers' feedback for 10 minutes. The relevant XP practices are *on-site customer* and *testing*.

The third and the fourth step are repeated twice (i.e., two iterations are performed).

4.3 Third Part

This part is entirely dedicated to teacher's feedback. Students are asked to present a "working prototype" in order to receive the last feedback before the presentation (during the fourth part). The teacher explains the assessment criteria (see Section 4.5) and provides specific feedback for each criterion. The relevant XP practices are *on-site customer* and *testing*.

4.4 Fourth Part

At the end of the activity, each student presents her/his set of slides in front of the class. The conformance with the initial requirements and mind map are checked. Therefore, relevant XP practices are *user stories* and *testing*. Moreover, the presentation provides an opportunity for peer feedback.

4.5 Assessment Criteria

For the evaluation of the final product (i.e., the set of slides), we consider the following three criteria:

1. structure: general organization of the slides;
2. content: correctness and completeness of the reported information;
3. citation of sources: this aspect is considered very relevant in K-12, in order to teach students the importance of giving credit to the creators of the original material.

Even if our didactic module focuses on the process of preparation of slides, one single didactic module does not allow us to assess the development of a "software engineering mindset". For this reason, for the process side, we limit our assessment to observing students' behaviour. These observations, indeed, will serve to the creation of an assessment framework for the process part in our future modules.

5 FIRST EDITION OF THE DIDACTIC MODULE

We have performed a first case study in a trilingual international high school in the city of Bolzano (Italy). For the CS field, this non-vocational high school includes in its curriculum:

- two hours per week of *ICT* in the first and second year, where students learn to evaluate, choose, and use different tools in order to solve a problem;

- two hours per week of *Computer Science* in the third, fourth and fifth year. At the final exam, students must show the ability to analyze and interpret data from different sources in order to explain a socio-economic event. Computer science lectures teach programming to achieve this goal.

Therefore, according to the classification provided by Ye and Fischer (Ye and Fischer, 2007), the students of this type of school are going to be end-users who program with a language specific to their application domain. This category, in the range introduced by Ye and Fischer, approaches the professional software developer; thus, there is a clear need to introduce software engineering principles into this school curriculum.

In our long-term vision, the peculiarity of this high school allows us to verify how fostering software engineering concepts by focusing only on the process side (which means, in the first two years) can benefit the students when they start programming (which means, during the third, fourth, and fifth year). For this reason, we have involved in our case study the first-year students (17 students - 12 F, 5 M), during the ICT lectures.

In the first year's ICT programme, one of the first topics in the syllabus is the "definition of information and communication technologies". In the perspective of learning-by-doing, in agreement with the ICT teacher, we have proposed the following activity: "prepare a presentation about ICT". The module was organized in four weekly blocks of 90 minutes, and students worked individually.

6 RESULTS

At the end of the didactic module, all the 17 students presented their set of slides about ICT, which all fulfilled the initial requirements. The number of ICT-related topics included in the presentations varied from student to student, ranging from a minimum of two topics to a maximum of four. The most covered topics in the presentations are listed in Table 1.

As detailed in Section 4.5, for product assessment we used three criteria: structure, content, and citation of sources. As shown in Figure 1, the structure of 12 presentations was sufficiently good, while 5 presentations were almost sufficient because of some issues, such as the absence of the title in some slides or the excessive presence of text. Six presentations had fully sufficient content and two were insufficient (i.e., the content was just superficially mentioned). Only three presentations did not cite the sources of information. In general, the final evaluation of the presentations was better than the evaluation obtained during the third lesson.

Regarding the process, it should be noted that the students did not understand the iterative model immediately. The majority of students, in fact, at the first request to submit a first result after 20 minutes tried to complete the entire assignment, without understanding that it was enough to submit a first version on which they could then have feedback to continue the rest of the work. Following the guidelines of the End-User Software Engineering field, we did not interrupt our activities to provide additional explanations; instead, we tried to let the students perceive (during the first iterations) the advantage of having feedback on their prototypes. After a first phase of "adjustment", we noticed that our students started to organize their activities in order to be able to present, during the next meeting with the teachers, the parts on which they needed more support. We consider this as an indicator that students started understanding the difference between a moment of evaluation and a moment of feedback.

If the iteration frequency appeared initially as something "stressful", at a later stage the students started asking for each task: "what time do we have to show a first prototype?". Finally, the iterative model has allowed even the slowest students to conclude the didactic module with a minimal, but still sufficient, presentation (i.e., two sub-topics).

The perceived usefulness of mind maps (implementation of user stories practice) emerged during a discussion with the students at the end of the activities. Students perceived mind maps as a support tool to avoid "getting lost on the Internet in the large amount of information" that they could find there. Therefore, the design phase of the presentation helped them to be more focused, while allowing them to change the design at a later stage to introduce new ideas.

Table 1: Topics covered in the presentations about ICT.

Topic	Number of presentations	% of presentations
ICT definition	12	70.6
Areas of application	11	64.7
Examples of applications in the educational environment	8	47.1
Historical notes	6	35.3
ICT objectives	5	29.4
Positive / negative aspects	5	29.4

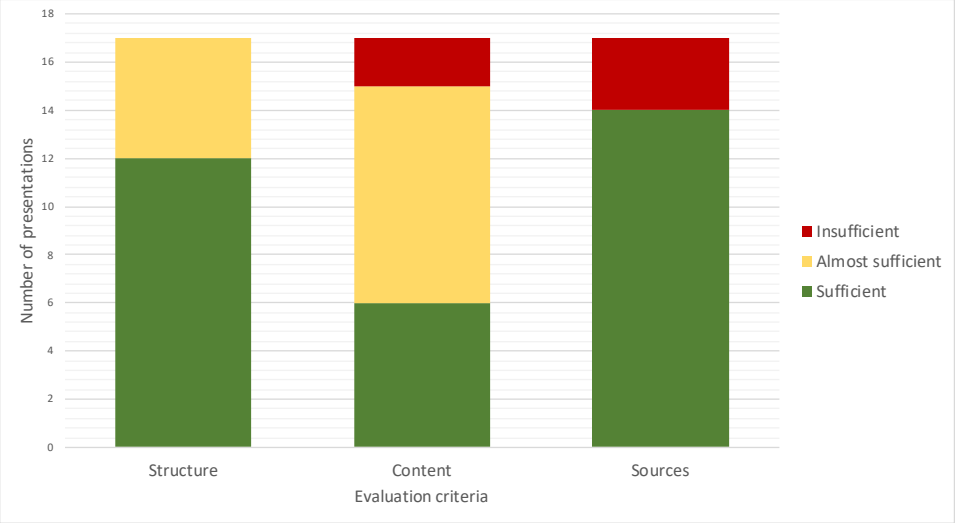

Figure 1: Assessment of the final product.

7 DISCUSSION AND CONCLUSION

In this work, we describe a didactic module in which we leverage a typical activity of the existing study programme (i.e., preparing a set of slide) in order to foster software engineering principles. The module does not shift the students' attention from their main objective and does not introduce additional lectures on software engineering. Paying attention to this last characteristic is of paramount importance, especially in non-vocational schools: in these schools, indeed, the goal is to foster a mindset that could be fundamental in the future career of students, who at the moment would not perceive additional lectures on SE as useful.

We performed a first classroom-based experimentation of this didactic module in a first class of an international trilingual high school. The first results show that, although no additional explanations were provided, students have adapted rather quickly to the new organization of the work process, by taking advantage of the frequent iterations and of the obtained feedback.

In the short term, we plan to design other modules that we will experiment with the same group of students. This will allow us to observe if and when students will develop a "software engineering mindset", even without developing software. The results of this effort could confirm the benefits of working in the EUSE perspective in different disciplines, including non-STEM ones. Moreover, further experiments would help in developing an assessment protocol for the process aspect, which is now only evaluated through observations. In future modules, we plan to include teamwork activities, which benefits the most of an XP approach, and to support them with specific practices (e.g., stand-up meeting).

In the long term, our goal is to verify how fostering software engineering concepts by focusing only on the process side can be beneficial to students when they start programming. Results in this direction would demonstrate the effectiveness of the activities carried out in creating a mindset to be applied also in the creation of software.

REFERENCES

Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Bollin, A., Pasterk, S., Antonitsch, P., and Sabitzer, B. (2016). Software engineering in primary and secondary schools-informatics education is more than programming. In *Software Engineering Education and Training (CSEET), 2016 IEEE 29th International Conference on*, pages 132–136. IEEE.

Burnett, M. (2009). What is end-user software engineering and why does it matter? In *International Symposium on End User Development*, pages 15–28. Springer.

Burnett, M. M. and Myers, B. A. (2014). Future of end-user software engineering: beyond the silos. In *Proceedings of the on Future of Software Engineering*, pages 201–211. ACM.

Chimalakonda, S. and Nori, K. V. (2013). What makes it hard to teach software engineering to end users? some directions from adaptive and personalized learning. In *Software Engineering Education and Trai-*

ning (CSEE&T), 2013 IEEE 26th Conference on, pages 324–328. IEEE.

Costabile, M. F., Mussio, P., Parasiliti Provenza, L., and Piccinno, A. (2008). End users as unwitting software developers. In *Proceedings of the 4th International Workshop on End-user Software Engineering*, WEUSE '08, pages 6–10, New York, NY, USA. ACM.

Fronza, I., El Ioini, N., and Corral, L. (2015). Students want to create apps: Leveraging computational thinking to teach mobile software development. In *Proceedings of the 16th Annual Conference on Information Technology Education*, SIGITE '15, pages 21–26, New York, NY, USA. ACM.

Fronza, I., El Ioini, N., Corral, L., and Pahl, C. (2018). *Agile and Lean Concepts for Teaching and Learning*, chapter Bringing the benefits of Agile techniques inside the classroom: a practical guide. Springer. To appear.

Fronza, I., El Ioini, N., Janes, A., Sillitti, A., Succi, G., and Corral, L. (2014). If i had to vote on this laboratory, i would give nine: Introduction on computational thinking in the lower secondary school: Results of the experience. *Mondo Digitale*, 13(51):757–765.

Fronza, I., Ioini, N. E., and Corral, L. (2017). Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education (TOCE)*, 17(4):19.

Kastl, P., Kiesmüller, U., and Romeike, R. (2016). Starting out with projects: Experiences with agile software development in high schools. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, pages 60–65. ACM.

Kropp, M. and Meier, A. (2013). Teaching agile software development at university level: Values, management, and craftsmanship. In *Software Engineering Education and Training (CSEE&T), 2013 IEEE 26th Conference on*, pages 179–188. IEEE.

Meerbaum-Salant, O. and Hazzan, O. (2010). An agile constructionist mentoring methodology for software projects in the high school. *ACM Transactions on Computing Education*, 9(4):n4.

Monteiro, I. T., de Castro Salgado, L. C., Mota, M. P., Sampaio, A. L., and de Souza, C. S. (2016). Signifying software engineering to computational thinking learners with agentsheets and polifacets. *Harvard Business Review*.

Steghöfer, J.-P., Knauss, E., Alégroth, E., Hammouda, I., Burden, H., and Ericsson, M. (2016). Teaching agile: addressing the conflict between project delivery and application of agile methods. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 303–312. ACM.

Wing, J. M. (2006). Computational thinking. *Comm. ACM*, 49(3).

Ye, Y. and Fischer, G. (2007). Designing for participation in socio-technical software systems. *Universal Acess in Human Computer Interaction. Coping with Diversity*, pages 312–321.