Teaching Software Engineering Principles in Non-vocational Schools

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

Keywords: Software Engineering Training and Education, End-user Software Engineering, K-12.

Abstract: Many activities, such as computational thinking courses, are nowadays proposed in K-12 to prepare students for the current labour market, where being able to creatively use technology to solve problems is becoming increasingly important, and where more and more people are engaged in programming activities. Thus, there is a need to equip students with the necessary means to improve software quality, including non-vocational schools, where the challenge is leveraging existing curricular, non-programming activities to this end. This work explores the possibility of fostering software engineering principles in non-vocational high schools through curricular, non-programming activities. We describe two didactic modules and report the results of a classroom experience (involving 16 high school first-year students) that has been carried out to understand the effectiveness of the proposed approach. During the didactic modules, the participants achieved the objectives of the curricular activity, and at the same time learned how to organize their work by applying software engineering principles. These results 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".

1 INTRODUCTION

Computational thinking can be defined as "the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer - human or machine - can effectively carry out" (Wing, 2014). 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, 2014). As a consequence, many activities are nowadays proposed in K-12 to foster computational thinking skills also in non-vocational schools, where students are generally less experienced in Science, Technology, Engineering and Mathematics (STEM).

The broader aim is to prepare students for the current labour market, where being able to creatively use technology to solve problems (rather than being passive users) is becoming particularly important. At the same time, outside the educational setting, the number of "unofficial" software developers is already considerable: the research based on U.S. Bureau of Census and Bureau of Labor data shows that there are about three million professional programmers in the United States, but over twelve million more people who say they do programming at work (Scaffidi et al., 2005). Even the term *end-user*, initially considered as an antonym of *professional developer*, encompasses now a wide range of software-related activities (Burnett and Myers, 2014) and refers also to unqualified people who produce software in the labor market.

The downside is the overall low quality of end-user-produced software (Burnett, 2009). This aspect is critical because, even if the errors are non-catastrophic, their effects can be serious. For example, a small-business owner might create a web application with the goal of promoting her business; nevertheless, this might result in a loss of revenue and credibility if, for example, this web application contains pages that are displayed incorrectly (Burnett, 2009). According to R. Scheubrein, one of the reasons of the overall low quality of end-user-produced software is that most of the end-users lack an explicit training in software engineering (Scheubrein, 2003).

If computational thinking activities in K-12 will be successful, in the near future we can expect more and more people to be engaged in programming activities, even without a specific training (Scaffidi et al., 2005) (Fronza et al., 2019). Therefore, in order to increase the quality of end-user-created software, it is crucial to equip students with the necessary means to improve software quality. This means that End-User Software Engineering (EUSE), which aims at bringing the benefits of a Software Engineering (SE) ap-

Fronza, I. and Pahl, C.
Teaching Software Engineering Principles in Non-vocational Schools.
DOI: 10.5220/0007672702520261
In *Proceedings of the 11th International Conference on Computer Supported Education (CSEDU 2019)*, pages 252-261
ISBN: 978-989-758-367-4

proach to end-users (Burnett and Myers, 2014), needs to be introduced also in the K-12 environment. Focusing on undergraduate students might be a good start, but can not be the ultimate goal when there is a need to reach all the students before they choose their career, which means in K-12–including non-vocational schools. Moreover, K-12 students can benefit from learning SE principles also in other fields: for example, being able to find a solution incrementally can be useful in many contexts outside the CS classroom (Fronza et al., 2019).

The endeavor to bring software engineering principles to K-12 sets a number of challenges. For example, end-users generally do not perceive the usefulness of learning SE principles, and are focused exclusively on solving their specific problem (Chimalakonda and Nori, 2013). In this respect, EUSE aims at respecting end-users goals and working habits, without aiming at transform them into professional software engineers through ad-hoc courses and activities (Burnett and Myers, 2014).

Based on these considerations, in this work we describe two didactic modules that we have designed to foster SE principles in non-vocational high schools, by leveraging non-programming activities that are already present in the curriculum. For this reason, the modules objectives are the creation of an *infographic* and of a *video*. The proposed activities have been designed to foster a "software engineering mindset" without introducing additional lectures on SE. In order to evaluate the effectiveness of the proposed modules, we have performed a classroom experience that involved 16 high school first-year students. The results of this experience show that the proposed modules allowed the participants both to reach the curricular objectives and to learn how to organize their activities by applying a SE approach. These results allow us to formulate hypotheses for further work, such as extending our approach to other activities and observe if students will develop a software engineering mindset, even without developing software.

The remaining part of the paper is organized as follows. Section 2 describes the state of the art of EUSE in primary and secondary schools; Section 3 describes the rationale of the proposed didactic modules, and Section 4 details their structure and the assessment strategy. Section 5 describes the classroom experience, and Section 6 details its results. Section 7 discusses our results and draws conclusions from this work, also proposing possible directions for future work.

2 STATE OF THE ART

According to Mary Shaw, software development should be treated from an engineering point of view (which means resolving constraints, considering users, comparing alternatives, and so on) not only while teaching to prospective software engineers, but also to all students who learn software development (Shaw, 2000). Based on this statement, large attention has been dedicated to this research topic, and Software Engineering and Education have been the focus of extensive research in the last years, especially at undergraduate level. This effort has laid down the recent foundation of the field of End-User Software Engineering, which aims at bringing the benefits of a Software Engineering (SE) approach to end-users, i.e., to people who develop software without having a CS training (Burnett, 2009).

The large number of activities that have been introduced in K-12 to foster coding and computational thinking skills has somehow shifted M. Shaw's challenge to K-12. In this respect, Bollin et al. underlined the need and analysed the 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, communication skills, logic, planning, modelling, and computational thinking as an ability to solve problems (Bollin et al., 2016).

Nevertheless, only a few End-User Software Engineering studies address the problem of teaching SE in primary and secondary schools. In particular, most of them explore the possibility of teaching Agile methods, which have been shown to be a good candidate to organise the software development process in K-12 (Fronza et al., 2019). Among these studies, Meerbaum and Hazzan focused on high schools and presented a mentoring methodology on Agile (Meerbaum-Salant and Hazzan, 2010); Kastl et al. applied Agile methods in class and achieved greater flexibility in software development projects (Kastl et al., 2016). In the context of middle schools, Fronza et al. proposed a framework in which a series of Agile practices have been adapted to the specific context (Fronza et al., 2017).

With this vision at hand, we explored further the possibility of fostering Software Engineering principles in K-12, by focusing on non-vocational high schools, where the EUSE challenges are amplified, because additional lectures on SE would not be perceived as "useful", especially by those students who are less experienced in STEM. Therefore, our goal is exploiting existing curricular, non-programming activities, without introducing specific lessons on SE.

The rationale behind this choice is that, following the EUSE guidelines (Burnett and Myers, 2014), we aim at fostering SE principles by respecting the students' goals (i.e., we do not shift students' attention from the goal of completing their curricular tasks) and without introducing additional lectures on Software Engineering.

3 RATIONALE

The endeavor to bring the benefits of a Software Engineering (SE) approach to end-users should take into account end-users' goals and working style (Burnett, 2009). To this purpose, Agile methods have been shown to be a good option, because they favor a flexible, iterative approach, and focus more on the product than on the production of unnecessary documentation (Fronza et al., 2019). This approach conforms with end-users working style, which is preferably opportunistic and incremental (Burnett and Myers, 2014), collaborative (Costabile et al., 2008), and proceeds by trial-and-error phases (Burnett and Myers, 2014).

Nevertheless, the attempts to introduce Agile in K-12 need to deal with a long waterfall development model tradition (Kropp and Meier, 2013; Kastl et al., 2016), which takes into consideration the final product more than the journey undertaken by the students to produce it (Steghöfer et al., 2016). The introduction of Agile in K-12 would allow students less experienced in STEM subjects to focus on the process side, and learn SE principles also in activities that do not have software development as main objective. Therefore, switching to an Agile environment could lead to an opportunity to introduce SE principles also in non-vocational schools (Fronza and Zanon, 2015), thus reaching a larger number of students.

The resulting challenge is understanding how to leverage existing curricular activities to foster SE principles, without introducing specific lessons, i.e., without shifting students' attention from their goal of completing a curricular task (Fronza and Pahl, 2018). Thus, our Research Questions are:

- **RQ1:** Is it possible to leverage non-programming curricular activities to foster SE principles in non-vocational high schools?
- **RQ2:** Is this approach effective?

In order to answer these research questions, in this work we describe two didactic modules that we have designed to foster SE principles in non-vocational high schools. The objectives of the proposed modules are the creation of an infographic and the creation of a video. Among all existing curricular activities that we have analyzed together with a group of school teachers, we have selected these two topics because we wanted the modules objectives to be transversal to many disciplines; this way, the proposed modules can be easily adapted to different types of schools. To this end, we also considered the pressing need for visual communication skills, which is relevant for all students regardless of the course of study: successfully navigating the web requires the ability of finding images, analysing them, evaluating them, applying them to a purpose, and producing them (Conner and Browne, 2013; Matrix and Hodson, 2014).

Moreover, the current generation of students has grown up in a "YouTube environment" (Morgan, 2013). Thus, videos and infographics are a good strategy to tap into students' interests and engage them to learn also those subjects they are not particularly interested in (Spires et al., 2012). This aspect is particularly relevant when considering non-vocational schools (Fronza et al., 2015).

Finally, we do not include software development tasks in the proposed modules, so that students can focus only on the process side, which is crucial, as above-mentioned, especially in non-vocational schools. However, even without requiring programming activities, creating videos and infographics allows us to bring SE principles into the classroom. Software engineers do not just write programs: they think in terms of satisfying needs and solving problems. Moreover, following a (software) engineering process means managing all the steps from initial customer inception to the release of the final product. Indeed, SE methodologies (e.g., Agile) originated outside software in the first place.

Among the Agile methodologies, we have adopted eXtreme Programming (XP), because it provides a set of lightweight practices to guide the development process (Mikre, 2010); moreover, XP is based on frequent small releases, which provides an opportunity for formative assessment.

Since in our modules we wanted to focus on the process aspect, we have selected the set of XP practices, which are recommended to be adopted together (Fronza et al., 2019) and map to a set of Agile principles:

- *Incremental development* is supported through small releases, frequent testing, and user stories as a basis for deciding what functionality should be included in a system increment. User stories are implemented as informal prototypes (e.g., storyboards), which describe requirements in a language that is understood both by the team and the customers. Small releases consist in the decomposition of software development activities in short iterations in order to obtain timely and continuous feedback.

- *Customer involvement* is supported through the continuous engagement of the customer for defining acceptance tests and providing feedback. In this regard, having the instructor playing the role of a customer (Steghöfer et al., 2016) can also help in fostering students' self-organization on their projects (Kastl et al., 2016).
- *Change* is embraced through regular system releases to customers and test-first development, refactoring, and continuous integration of new functionality.

The following section details the structure of the proposed didactic modules and the assessment strategy.

4 STRUCTURE OF THE MODULES

The proposed modules cover a total of 28 hours of curricular activity (14 hours for each module).

4.1 First Module: Creation of an Infographic

An infographic is a representation of information in a graphic format designed to make the data easily understandable at a glance (an example is shown in Figure 1). In order to reach this goal, an infographic assignment challenges students to visually communicate a thesis, supported by citations and statistics sourced from the literature and the popular press. Moreover, it requires critical content analysis and filtering skills (Matrix and Hodson, 2014).

As shown in Table 1, this module covers 14 hours in total and includes the following activities.

4.1.1 Introduction to Infographics

The module begins with an introduction to infographics: tips are given for creating effective infographics, such as finding a clear narrative, visualizing information, choosing colors and fonts, and so on. Some examples are analysed together with students, in order to check their understanding of the provided tips. During this introductory presentation the focus is kept on infographics, and no notion is given neither about Agile methods nor about SE in general.

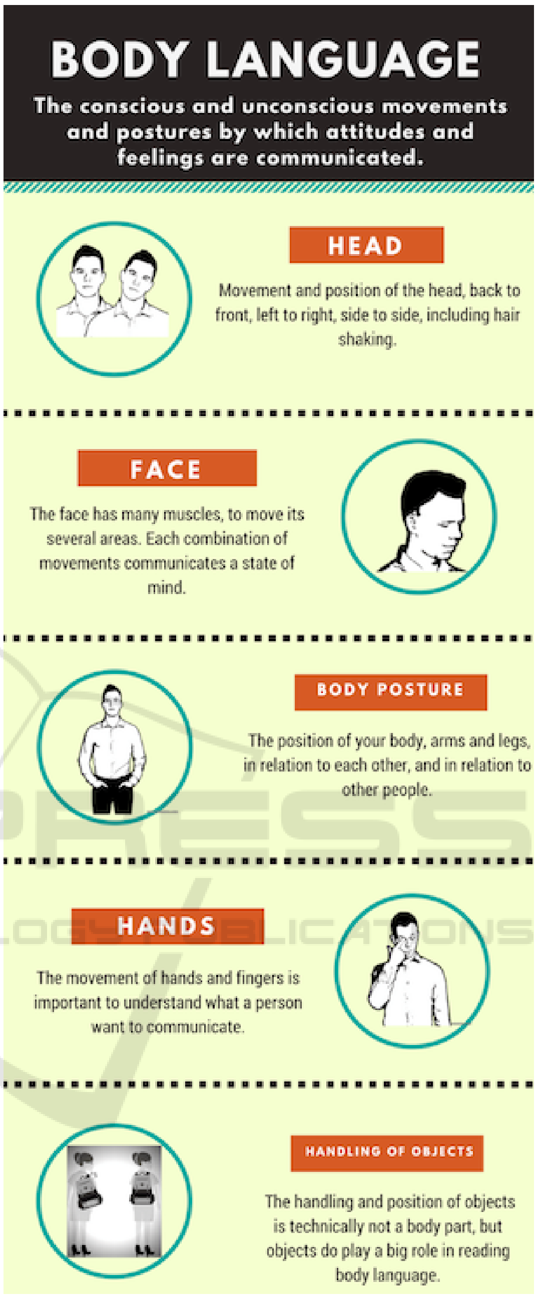
BODY LANGUAGE
The conscious and unconscious movements and postures by which attitudes and feelings are communicated.
HEAD
Movement and position of the head, back to front, left to right, side to side, including hair shaking.
FACE
The face has many muscles, to move its several areas. Each combination of movements communicates a state of mind.
BODY POSTURE
The position of your body, arms and legs, in relation to each other, and in relation to other people.
HANDS
The movement of hands and fingers is important to understand what a person want to communicate.
HANDLING OF OBJECTS
The handling and position of objects is technically not a body part, but objects do play a big role in reading body language.

Figure 1: An example of a student-created infographic.

4.1.2 Creation of a Paper-based Prototype

Students retrieve information on the central topic of the infographics, and then create a paper-based prototype to better organize ideas within the topic. To this purpose, students need to identify, for example, the structure of the infographic, the information flow, and the visualization means (e.g., symbols).

The activity is organised in iterations of twenty

Table 1: Struture of the first module: activities, hours covered, and XP practices that are fostered by each activity.

Activity	Hours	Practices
Introduction to infographics	2	
Creation of a paper-based prototype	6	User stories, on-site customer, small releases
Creation of the infographic	4	On-site customer, small releases, testing
Presentation in front of the class	2	User stories, testing
Total	14	

minutes, in which the relevant XP practice is *user stories*. At the end of each iteration, the current prototype is revised together with teachers. This activity requires 10 minutes, and fosters the following XP practices: *on-site customer, small releases*.

4.1.3 Creation of the Infographic

The infographic is created using Canva[1] starting from the paper-based prototype that has been produced during the previous phase. As in the previous part, students' work is organised into 20-minute iterations in order to foster the *small releases* XP practice. At the end of each iteration, teachers provide feedback for 10 minutes, which exercises the *on-site customer* and *testing* practices.

Before the last iteration, a peer evaluation model is also applied: students work in pairs and review the other's infographic, so that they can reflect on the feedback received from classmates and fix the infographic accordingly (Matrix and Hodson, 2014) (i.e., *testing* practice).

4.1.4 Presentation in Front of the Class

At the end of the module, each student presents her infographic in front of the class. The conformance with the initial requirements and paper-based prototype is checked. The relevant XP practices are *user stories* and *testing*. Moreover, this activity provides an opportunity for peer feedback.

Table 1 provides a summary of the practices that are fostered by each activity.

4.2 Second Module: Video Production

Creating a video engages the current "YouTube generation" of students (who frequently use videos to communicate and express themselves (Spires et al., 2012)) to learn also those topics they do not prefer (in our case, STEM in non-vocational schools). Moreover, it requires students to problem solve, think critically, and apply knowledge from a content area (Bell, 2005). Additionally, while creating a video students need to consider the ethical aspects of making a video, which includes asking for copyright permission (when necessary) and giving credit to others when using their ideas or help (Morgan, 2013). As shown in Table 2, this module covers 14 hours in total and includes the following activities.

[1] www.canva.com

Table 2: Struture of the second module: activities, hours covered, and XP practices that are fostered by each activity.

Activity	Hours	Practices
Introduction to video making	2	
Creation of a storyboard	4	User stories, on-site customer, small releases
Video production	6	On-site customer, small releases, testing
Presentation in front of the class	2	User stories, testing
Total	14	

4.2.1 Introduction to Video Making

The module begins with an introduction to the fundamentals of video making (e.g., video concept, scenes, framing, light, etc.). The same as in the infographics module, the focus is kept on video making, and no notion is given about software engineering or Agile.

4.2.2 Creation of a Storyboard

In this part of the module, students retrieve information about the central topic of the video (for the first 2 hours), which is decided by the teacher and is the same for the entire class. As a requirement, the video must have a maximum duration of 1.5 minutes (plus outro). Then, each student starts the pre-production phase of the video, by thinking about its objective and idea. This includes deciding the specific point of view on the given topic, story, scenes, framing and environments needed, etc. The design of the video is summarized in a paper-based storyboard.

The activity is organised in iterations of twenty minutes; at the end of each iteration, the current prototype is revised together with teachers (10 minutes). Therefore, this part of the module fosters the following XP practices: *user stories, on-site customer, small releases*.

4.2.3 Video Production

In order to create the video, students can use a tool of their choice; the only requirement is using their mobile phone throughout the entire production phase. The adoption of a Bring Your Own Device (BYOD) approach has been chosen in order to "demonstrate the potential of digital tools and encourage active and purposeful device use, to help learners overcome a more passive consumerist use of technology" (European Commission, 2016). Students work autonomously, but they are allowed to collaborate to the creation of others' videos (for example, as actors). Video shooting can happen also outside school hours. In order to exercise the *small releases, on-site customer* and *testing* practices, activities are organised into 20-minute iterations interleaved with 10-minute feedback sessions.

4.2.4 Presentation in Front of the Class

At the end of the module, each student shows the video to the class, and during the presentation the conformance with the initial requirements and storyboard is checked. The relevant XP practices are *user stories* and *testing*.

Table 2 provides a summary of the practices that are fostered by each activity.

4.3 Assessment Criteria

The two modules described in the first part of this section show how it is possible to leverage non programming curricular activities in order to foster SE principles (see RQ1). To answer our resarch question RQ2 we have defined an assessment strategy to evaluate the effectiveness of the proposed modules. The effectiveness of the modules depends on the extent to which they allow students to achieve two objectives. First, organizing the activity based on SE principle needs to guarantee the achievement of the curricular objectives. This is essential for the proposed modules to be successfully incorporated into the curriculum. To this end, we defined the following three assessment criteria for the product:

1. technical aspects, i.e., the conformity of the product with the desired characteristics, which are those listed in the modules introduction. For example, we checked: fonts and colors in infographics; length and image quality in videos;
2. narrative, i.e., the product follows a clear and logical line that guides the viewer from an introduction to a conclusion;
3. data representation, i.e., how visual tools (i.e., symbols, charts, framings) have been produced and used to convey the intended message.

The second component of the modules effectiveness is their ability to teach how to organize the proposed activities by applying SE principles. Thus, for this aspect the process side needs to be analysed. In this case, we limit our assessment to observing students' behaviour. A couple of modules do not allow us to assess the development of a "software engineering mindset", thus through these modules we collect observations in order to create an assessment framework for future modules. In order to assess the learnign of the XP practices, we observed the following:

1. *small releases*: positive assessment is given if students organize the activities in order to be able to present a prototype at the end of each iteration;
2. *on-site customer*: positive assessment is given if students take advantage of the teacher's (i.e., customer's) feedback to improve the product during the following iteration;
3. *testing*: positive assessment is given if students test/check the product frequently;
4. *user stories*: positive assessment is given if students use the paper-based prototype to guide the production process and, during the final presentation, are able to justify any changes to the initial prototype.

Maximum 2.5 points are awarded for each criterion and students achieve a sufficient assessment with 6 points.

The following section describes the first classroom experience.

5 CLASSROOM EXPERIENCE

We have performed a classroom experience in a non-vocational high school in Italy. The curriculum of this non-vocational high school includes the following subjects in the CS area:

- two hours per week of *ICT* during the first two years, where students learn to evaluate, choose, and use different tools in order to solve a problem;
- two hours per week of *Computer Science* from the third to the fifth year, where basic programming concepts are taught to support a specific goal, i.e., being able to analyze and interpret data from different sources in order to explain an event.

According to the classification provided by Ye and Fischer, the students of this type of school can be compared to those end-users who program with a language specific to their application domain (Ye and Fischer, 2007); thus there is a clear need to introduce SE principles into this curriculum.

In our long-term vision, the peculiarity of this high school will allow us to verify how fostering SE principles by focusing only on the process side (which means, in the first two years) can benefit the students when they start programming (which means from the third to the fifth year). For this reason, we have involved the first-year students (16 students - 11 F, 5 M) in our classroom experience.

One of the authors of this paper has worked alongside the ICT teacher during the entire school year; therefore, there was a high degree of interaction of the researcher with the subjects being studied but, at the same time, students had low awareness of being observed for research purposes (Wohlin et al., 2000), because the researcher was introduced as a teacher.

The two modules have been organized in weekly blocks of two school hours, and students worked individually. Before running the two modules presented in this paper, we performed a "pilot study" with the same group of students (plus one student who has only been attending the school for a short period of time). In that case, we focused on the process side by fostering XP practices through the preparation of a set of slides for a presentation (Fronza and Pahl, 2018).

Together with the school teacher, we chose as a theme of the two modules *non-verbal communication*, which is one of the curricular topics. In particular, infographics focused on *gestures*, while videos focused on *body language*. Students could choose to cover a specific aspect within these topics.

Both for infographics and videos, we suggested to students to produce the necessary material themselves (e.g., pictures of gestures, frames, etc.) or to give credit to others when using their ideas, material, or help.

6 RESULTS

As detailed in Section 4.3, our assessment included both product and process aspects. In the following, we report the results of the assessment of these two aspects.

6.1 Product Assessment

At the end of the **first module**, all the 16 students presented an infographic that fulfilled the initial requirements, i.e., covering an aspect of the theme *gestures*. In particular, the following aspects have been tackled: 1) examples of the five types of gestures (i.e., emblematic, iconic, metaphoric, affect, and beat), 2) meaning of a set of gestures, 3) gestures by age, and 4) gestures by country.

According to the assessment criteria detailed in Section 4.3, 12 infographics resulted to be sufficient (two of them were excellent), 3 were nearly sufficient, and 1 infographic was insufficient. Figure 2 shows the assessment of each criterion.

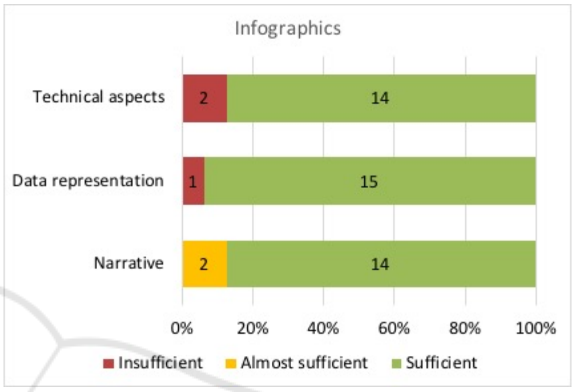

Figure 2: Product assessment at the end of the first module.

At the end of the **second module**, only 15 students delivered their video; one student, in fact, moved to another school and was therefore no longer participating in the activities.

Within the theme *body language*, the topics of the videos can be divided into two main categories: 1) tutorials about body language, in different situations and contexts (e.g., a tv show, or two students studying together), and 2) friendships in which communicating is difficult and therefore body language is used to communicate (e.g., two friends who do not speak a common language, or one mute friend).

According to the assessment criteria detailed in Section 4.3, 12 products were sufficient, 2 nearly sufficient, and 1 insufficient. Figure 3 shows the assessment of each criterion.

In both modules, we observed a quality improvement of the product throughout the iterations. Moreover, both in their infographics and videos, students gave credit to others when using their ideas or help, thus showing an understanding of this ethical aspect.

Altogether, these results indicate the effectiveness of the two didactic module in allowing the participants to achieve the curricular objectives (see RQ2). This means that the proposed approach does not undermine the achievement of the objectives of the discipline in which the software engineering approach is introduced.

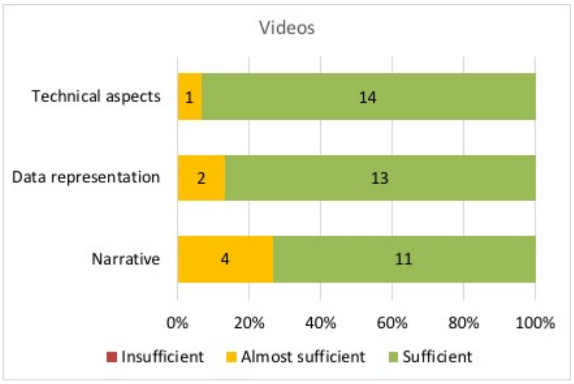

Figure 3: Product assessment at the end of the second module.

6.2 Process Assessment

Based on the process assessment criteria (described in Section 4.3), at the end of the first module ten students obtained a sufficient assessment. At the end of the second module, this number increased to fourteen. These students understood that we were not working with a waterfall approach, and organized their activities in order to be able to present a prototype at the end of each iteration to get feedback (i.e., *small reseases* practice), which they started to perceive as useful for reaching their objective (*on-site customer* practice). Here it should be noted that, probably, the first "pilot study" (see Section 5) already contributed to training them on the concept of iterative processes.

Presenting a prototype at the end of each iteration prompted students to test/control the prototype frequently, so as to avoid making a bad impression (*testing* practice). For this reason we think that an assessment framework should check the correct "functioning" of the solution at each iteration, which is something we have not kept track formally.

In the second module, we observed an increase (from 10/16 to 14/15) of the number of students who started using paper-based prototypes (*user stories* practice) as a support during the creation of the final product (an example is shown in Figure 4).

In many cases, students changed their plan respect to the initial paper-based prototype, for example because they did not find the needed data, or did not have a sufficient number of actors available. Those students who at the end got an insufficient assessment changed the product during the production phase (in an iterative fashion), but did not update the design accordingly (in a waterfall fashion), even if they did have the chance (and were encouraged) to adapt it. As a result, since they overlooked this aspect, during the final presentation they could not explain to the client

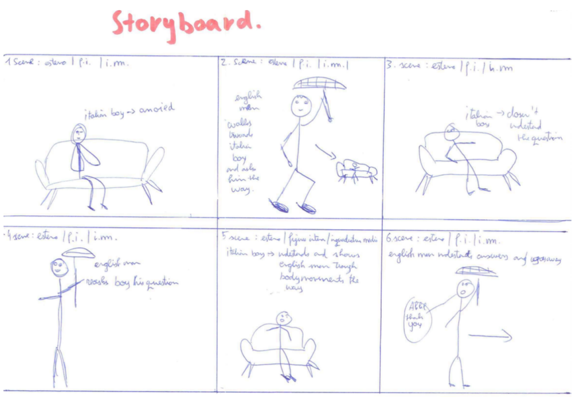
Storyboard.
1 Scene : estero | p.i. | i.m.
italian boy → annoied
2. Scene : estero | p.i. | i.m. |
english man
walks toward italian boy and asks him the way.
3. scene : estero | p.i. | h.m
italian boy → doesn't understand the question
4 scene : estero | p.i. | i.m.
english man reasks boy his question
5 scene : estero | figure intere | inquadratura media
italian boy → understands and shows english man trough bodymovements the way
6. scene : estero | p.i. | i.m.
english man understands answers and goes away
thank you

Figure 4: An example of storyboard created during the second module.

why the final product differed from the promised one.

These results 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.

7 CONCLUSION AND FUTURE WORK

With the goal of understanding if it is possible to leverage non-programming curricular activities to foster Software Engineering (SE) principles in non-vocational high schools (RQ1), this work described two didactic modules that have been designed for this purpose. The modules do not shift the students' attention from their main objective (i.e., completing their curricular tasks) and do not introduce additional SE lectures. Paying attention to this last characteristic is particularly important, especially in non-vocational schools: in these schools, the goal is to foster a SE mindset that could be fundamental in the future career of students, who at the moment would not perceive additional lectures on SE as useful.

The results of the classroom experience reported in this paper indicate the effectiveness (RQ2) of the two didactic module in allowing the participants to achieve the objectives of the discipline in which the SE approach was introduced; moreover, although no additional explanations were provided, students adapted to the new organization of the work process, by taking advantage of the frequent iterations and of the obtained feedback.

Further experiments are needed to confirm these results and to develop a comprehensive assessment protocol for the process aspect. However, these findings can be taken as an indicator of the benefits of working in the EUSE perspective in different disciplines, including non-STEM ones.

In future modules, we plan to include teamwork activities, which benefit 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 SE practices through non-programming activities (i.e., by focusing only on the process side) can be beneficial to students when they start programming.

ACKNOWLEDGEMENTS

The authors would like to thank all the students and their teacher for taking part to the classroom experience described in this paper.

REFERENCES

Bell, A. (2005). Creating digital video in your school. *Library media connection*, 24(2):54.

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

Conner, M. and Browne, M. (2013). Navigating the information-scape: information visualization and student search. *Reference Services Review*, 41(1):91–112.

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.

European Commission (2016). Bring your own device (byod). Technical report, ET 2020 Working Group on Digital Skills and Competences, European Commission.

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. (2019). *Agile and Lean Concepts for Teaching and Learning*, chapter Bringing the benefits of Agile techniques inside the classroom: a practical guide, pages 133–152. Springer.

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.

Fronza, I. and Pahl, C. (2018). End-user software engineering in K-12 by leveraging existing curricular activities. In *Proceedings of the 13th International Conference on Software Technologies, ICSOFT 2018, Porto, Portugal, July 26-28, 2018.*, pages 283–289.

Fronza, I. and Zanon, P. (2015). Introduction of computational thinking in a hotel management school [introduzione del computational thinking in un istituto alberghiero]. *Mondo Digitale*, 14(58):28–34.

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.

Matrix, S. and Hodson, J. (2014). Teaching with infographics: Practicing new digital competencies and visual literacies. *Journal of pedagogic development*, 4(2).

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.

Mikre, F. (2010). The roles of assessment in curriculum practice and enhancement of learning. *Ethiopian Journal of Education and Sciences*, 5(2).

Morgan, H. (2013). Technology in the classroom: Creating videos can lead students to many academic benefits. *Childhood Education*, 89(1):51–53.

Scaffidi, C., Shaw, M., and Myers, B. (2005). Estimating the numbers of end users and end user programmers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 207–214. IEEE.

Scheubrein, R. (2003). Elements of end-user software engineering. *INFORMS Transactions on Education*, 4(1):37–47.

Shaw, M. (2000). Software engineering education: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 371–380. ACM.

Spires, H. A., Hervey, L. G., Morris, G., and Stelpflug, C. (2012). Energizing project-based inquiry: Middle-grade students read, write, and create videos. *Journal of Adolescent & Adult Literacy*, 55(6):483–493.

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. (2014). Computational thinking benefits society.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

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