Informatics in Education, 2018, Vol. 17, No. 2, 167–206

DOI: 10.15388/infedu.2018.10

Teaching Software Engineering in K-12 Education: A Systematic Mapping Study

Fernando da CRUZ PINHEIRO,
Christiane Gresse von WANGENHEIM, Raul MISSFELDT FILHO
Department of Informatics and Statistics (INE) – Federal University of Santa Catarina (UFSC), Brazil
e-mail: fernando.pinheiro@posgrad.ufsc.br, c.wangenheim@ufsc.br, raul.missfeldt.filho@grad.ufsc.br

Received: July 2018

Abstract. Diverse initiatives have emerged to popularize the teaching of computing in K-12 mainly through programming. This, however, may not cover other important core computing competencies, such as Software Engineering (SE). Thus, in order to obtain an overview of the state of the art and practice of teaching SE competences in K-12, we carried out a systematic mapping study. We identified 17 instructional units mostly adopting the waterfall model or agile methodologies focusing on the main phases of the software process. However, there seems to be a lack of details hindering large-scope adoption of these instructional units. Many articles also do not report how the units have been developed and/or evaluated. However, results demonstrating both the viability and the positive contribution of initiating SE education already in K-12, indicate a need for further research in order to improve computing education in schools contributing to the popularization of SE competencies.

Keywords: software engineering, teaching, K-12.

1. Introduction

Currently, the introduction of teaching computing in schools is a worldwide trend (Hubwieser, 2012) supported by a number of initiatives such as Code.org (Code, 2018), Code Club (CodeClubWorld, 2018), Girls Who Code (Girlswhocode, 2018), Black Girls Code (Blackgirlscode, 2018) or Computação na Escola (CNE, 2018), among others. These initiatives aim to teach computational thinking (Wing, 2008) as an important 21st century skill, as well as to spark students' interest in STEM (Science, Technology, Engineering, and Mathematics) and IT (Information Technology). Most of these initiatives focus specifically on coding exercises as the main curriculum subject, using age-appropriate block-based programming environments such as Scratch (Scratch, 2018).

However, this approach mainly focused on teaching programming, may not cover other important computing core competencies, such as Software Engineering (SE), which are essential for the development of software (ACM/IEEE, 2013). These competencies are rarely covered in computing courses in schools, even in high school (Verhoeff, 2006). The inclusion of SE practices with respect to requirements development, software design, user interface design, testing, configuration management, etc. can help students gain insight into some of the challenges in real software projects (ACM/IEEE, 2013). Learning how to program can not be separated from SE. To create software, students need to learn at least the basic steps of the software process (Bollin and Sabitzer, 2015). Even in a regular programming course, it is important to address the basics of software engineering, being helpful to prevent and correct mistakes and, thus, increasing the joy of programming. On the other hand, teaching programming without attention for SE can make programming unnecessarily harder and more frustrating, especially when getting involved in more challenging programming assignments. SE education is responsible for a broad spectrum of competencies that software engineers need for their professional life. Being able to produce software in a systematic, controlled and efficient manner in a variety of contexts requires an extensive knowledge on a range of models, methods and tools from different SE knowledge areas, together with the understanding necessary to select and deploy them (ACM/IEEE, 2014). Typically, these competencies are taught in higher education in undergraduate or graduate computing courses (Shackelford *et al.*, 2005). So, considering the trend to introducing computing education already in K-12 in order to popularize computing, it also becomes important to introduce the learning of SE competencies at this educational stage (Bollin and Sabitzer, 2015). Yet, as it is of course not feasabile to cover all SE topics on the same level of detail as in higher education due to curriculum objectives and constraints (Bollin and Sabitzer, 2015), the question that arises is whether and how SE is taught today in K-12.

So far, several articles present overviews on the teaching of computing. Grover and Pea (2013) present a systematic review on the teaching of computational thinking in K-12. Several authors also provide global views on how teaching computer science is approached by several countries in K-12 (Hubwieser *et al.*, 2015; Heintz *et al.*, 2016). Some reviews provide an overview on the adoption of programming environments such as Scratch (Moreno-León and Robles, 2016) and/or specific types of applications (e.g. robotics) (Bascou and Menekse, 2016). However, none of these reviews specifically addresses the teaching of SE competencies in K-12. On the other hand, several reviews analyze the state of the art of teaching Software Engineering in general (Malik and Zafar, 2012; Shaw, 2000; Mead, 2009) or specific SE topics, such as processes (Heredia *et al.*, 2015). Other reviews related to SE teaching focus on specific instructional methods, such as games (Kosa *et al.*, 2016; Gresse von Wangenheim and Shull, 2009). However, these reviews focus exclusively on SE education in higher education.

Thus, in order to analyze the question of whether and how SE teaching is approached in schools, we carry out a systematic mapping study to identify, select, classify and analyze published studies. The main contribution of this article is the mapping and synthesis of the characteristics of instructional units (IUs) for SE education in K-12, regarding

their content, context and the analysis of how they were developed and evaluated. Our results show that it may be possible and beneficial to introduce SE education in K-12. In addition, the overview can help instructors select and/or develop instructional units in order to integrate teaching SE into their classes, as well as guide curriculum developers. It can also help instructional researchers and designers to improve IUs identifying improvement opportunites. We also hope that the discussion can further foster the inclusion of SE education in K-12.

2. Background

2.1. *Software Engineering*

Software Engineering (SE) is a knowledge area of computing that defines systematic, disciplined and quantifiable approaches for the development, operation and maintenance of software (IEEE, 2010). SE involves several knowledge areas as presented in Table 1.

Table 1
SE Knowledge Areas (IEEE CS, 2014)

Knowledge area	Description
Software Requirements	Area concerned with the elicitation, analysis, specification, and validation of software requirements as well as the management of requirements during the whole life cycle of the software product
Software Design	The process of definition of the architecture, components, user interfaces, and other characteristics of a system or component and the result of process
Software Construction	Refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging
Software Testing	Consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain
Software Maintenance	Refers to all activities required to provide an economically viable software support, covering various techniques (reengineering, reverse engineering, etc.)
Software Configuration Management	A life cycle support process that benefits project management, development, maintenance and quality assurance activities, as well as customers and users of the final product. The process covers the identification, control, status documentation, software configuration auditing as well as the management and delivery of software deliverables
Software Engineering Management	Refers to software management activities, such as planning, coordination, measurement, monitoring, control and documentation, to ensure that software products and services are delivered in an effective and efficient manner, with desired quality and that benefit the stakeholders
Software Engineering Process	Software engineering processes are concerned with work activities accomplished by software engineers to develop, maintain, and operate software, such as requirements, design, construction, testing, configuration management, and other software engineering processes

Continued on next page

Table 1 – continued from previous page

Knowledge area	Description
Software Engineering Models and Methods	This knowledge area emphasizes on software engineering models and methods that encompass multiple software life cycle phases, since methods specific for single life cycle phases are covered. The models provide an approach to problem solving, a notation, and procedures for model construction and analysis. Methods provide an approach to the systematic specification, design, construction, test, and verification of the end-item software and associated work products
Software Quality	This area addresses definitions and provides an overview of practices, tools and techniques for software quality assurance, control and quality assessment during development, maintenance, and deployment
Professional Practice of Software Engineering	This knowledge area is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner
Software Engineering Economics	Software engineering economics is about making decisions related to software engineering in a business context
Key Areas Important to Software Engineering	
Computing Foundations	This knowledge area encompasses the development and operational environment in which software evolves and executes. Because no software can exist in a vacuum or run without a computer, the core of such an environment is the computer and its various components
Mathematical Foundations	This area covers basic techniques to identify a set of rules for reasoning in the context of the system under study.
Engineering Foundations	This area outlines some of the engineering foundational skills and techniques that are useful for a software engineer. The focus is on topics that support other knowledge areas while minimizing duplication of subjects covered elsewhere in this document

For each of these knowledge areas, the Software Engineering discipline provides a number of processes, models, methods and techniques, as well as tools and notations.

2.1.1. *Teaching Software Engineering*

Currently, no curriculum guides exist that are specifically aimed at teaching Software Engineering in K-12. However, in general, based on the SE 2014 curriculum guide for higher education (SEEK) (ACM/IEEE, 2014), it is expected that students in higher education will be able to demonstrate the following competencies:

- Professional knowledge: Show mastery of software engineering knowledge and skills and of the professional standards necessary to begin practice as a software engineer.
- Technical knowledge: Demonstrate an understanding of and apply appropriate theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, verification, and documentation.

- Teamwork: Work both individually and as part of a team to develop and deliver quality software artifacts.
- End-user Awareness: Demonstrate an understanding and appreciation of the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment.
- Design Solutions in Context: Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal, and economic concerns.
- Perform Trade-Offs: Reconcile conflicting project objectives, finding acceptable compromises within the limitations of cost, time, knowledge, existing systems, and organizations.

SE 2014 SEEK (ACM/IEEE, 2014) based on SWEBOK (IEEE CS, 2014), defines that SE education at the undergraduate level should address the following SE knowledge areas:

- Requirements Analysis and Specification.
- Software Modeling and Analysis.
- Software Verification and Validation.
- Software Process.
- Software Quality.
- Security.
- Professional Practice.
- Computing Essentials.
- Mathematical and Engineering Fundamentals.

Following SE 2014 SEEK (ACM/IEEE, 2014), students at the undergraduate level should learn competencies according to Bloom's taxonomy (Bloom, 1956) at the cognitive levels of: knowledge (remembering previously learned material), comprehension (understanding information and the meaning of material presented), and application (using learned material in new and concrete situations).

These curriculum guides for higher education, thus, may indicate SE competencies also relevant to SE education for K-12, yet, with a reduced scope and/or aiming at lower levels of Bloom's taxonomy also depending on the specific type of school (such as technical schools).

2.2. *Computing Education in K-12*

K-12 education is basically composed of preschool, primary and secundary education (Table 2)(US Departament Education, 2018).

Currently, computing education in K-12 is often carried out as an extracurricular activity in the form of clubs, summer camps, workshops, etc. There are also several Massive Online Open Courses (MOOCs) available online specifically aimed at K-12 (Heintz *et al.*, 2016; Hermans and Aivaloglou, 2017). Computing education has also been included in several countries in the K-12 curriculum, teaching computing as an independent discipline and/or by integrating the content in a multidisciplinary way in oth-

Table 2
K-12 educational stages (US Departament Education, 2018)

Age	Educational stage	3-stage system
4	Early childhood education	Preschool
5	Primary education	Elementary school
6		
7		
8		
9		
10		Middle school
11		
12	Secondary education	
13		
14		
15		
16		High school
17		
18		

er disciplines throughout the curriculum (Heintz *et al.*, 2016). According to the CSTA, K–12 Computer Science Framework (2017) teaching computing in K-12 should address several core concepts and practices as presented in Table 3.

A major focus in this context is on teaching algorithms and programming by teaching students to program various types of software such as games, animations or mobile applications (Lye and Koh, 2014). Therefore, typically block-based programming environments, such as Scratch (MITb, 2018), Alice (Alice, 2018) or App Inventor (MITa, 2018) are used with novice students. Block-based programming languages motivate the learning of programming concepts by focusing on the logic and structures involved in programming, not requiring the learning of complex syntax and semantics as necessary in textual programming languages (Grover *et al.*, 2015). These environments also allow students to develop programs more easily, as results typically can be tested and viewed immediately in the form of animations, games or mobile applications.

Table 3
Core concepts and practices in computer teaching (CSTA, 2017)

Core Concepts	Core Practices
• Computer Systems	• Promote an inclusive computing culture
• Networking and Internet	• Collaborate on computing
• Data and Analysis	• Recognize and define computational problems
• Computer Impacts	• Develop and use abstractions
• Algorithms and Programming	• Create computational artifacts
	• Test and refine computational artifacts
	• Communicating about computing

This allows students to acquire problem solving skills by applying the engineering development cycle in practice (Lye and Koh, 2014). Advancing computing education, commonly text-based programming languages such as as Java, Python or C++ are introduced typically at high school level (Hubwieser *et al.*, 2015). Another popular approach is teaching robotics technologies such as the Lego Robot programming language (Starrett, 2007) or PBASIC (Corbett and Nesiba, 2015), a microcontroller-based version of BASIC.

For computing education in K-12 diverse instructional methods are used varying from direct instruction (e.g., lectures) to independent studies (Table 4) (Saskatchewan Education, 1991).

In accordance to learning objectives aiming at teaching the application of algorithm and programming concepts, a predominance of active learning strategies is observed, which allows the student to apply the competences to be learned. These strategies include exercises, such as developing code for a well-defined problem as well as the adoption of constructivist approaches such as situated learning, project-based learning, among others, dealing with open-ended and ill-defined problems. According to the instructional strategies adopted, several types of instructional materials are used, including software artifacts (e.g., specification of requirements of a predefined software system, use cases, user stories, code samples), exercise sheets, slides, videos, examples, among others (Lye e Koh, 2014). The assessment of the student's learning is usually done by the instructor using diverse methods, such as observations, questionnaires, interviews, etc. For assessing programming assignments, typically, performance-based assessments are adopted by manually or automatically analyzing the artifacts (e.g., software) created by the students (Gresse von Wangenheim *et al.*, 2018). In the context of game-based learning approaches, the scores of the game may also be used for assessment (Rusu *et al.*, 2011). Peer assessment is another way, in which the artifacts created by the students are assessed by their own peers (De Kereki and Manataki, 2016).

Table 4
Instructional Methods (Saskatchewan Education, 1991)

Direct Instruction	Indirect Instruction	Interactive Instruction	Independent Study	Experimental Learning
• Structured Overview • Explicit Teaching • Lecture • Drill and Practice • Compare and Constrast • Didactic Question • Demonstration • Guides for Reading, Listening, Viewing	• Case Study • Problem Solving • Inquiry • Reading for Meaning • Reflective Discussion • Concept Formation • Concept Mapping • Concept Attainment • Cloze Procedure	• Debate • Role Playing • Brainstorming • Panel • Peer Practice • Discussion • Laboratory Group • Co-operative Learning Group • Problem Solving • Circle of Knowledge • Tutorial Groups • Interviewing • Contests	• Essay • Computeer Assisted Instruction • Reports • Learning Activity Package • Correspondence Lessons • Learning Contracts • Homework • Research Projet • Assigned Question • Learning Centre	• Field Trip • Conducting Experiment • Simulation • Focused Imaging • Game-based learning • Field Observation • Role Playing • Synectics • Model Building • Survey

2.3. *Development and Evaluation of Instructional Units*

Instructional units (courses, workshops, etc.) are typically developed in a systematic way using instructional design (Branch, 2009), in order to make the acquisition of competencies more efficient, effective, and appealing. Instructional design defines an iterative process of planning learning objectives, selecting instructional strategies (including the design of assessments), selecting or creating instructional material, and applying and evaluating IUs (Branch, 2009).

Among instructional design models, one of the most popular models is ADDIE (Branch, 2009), including the following phases: During the analysis phase, the learning needs are identified. As part of the analysis the goals and objectives of the instructional unit are determined and the target audience is characterized. Other factors, such as human and technical resources, infrastructure, cost and time, etc., are analyzed. During the design phase, the learning objectives of the IU are specified. The content to be addressed is defined and sequenced, and the instructional methods to be used are defined. It is also defined how the students' learning will be assessed. As a result, the sylabus is defined. During the development phase, the material that will be used during the instructional unit is selected and/or created in accordance to the defined instructional methods. This step may also involve the selection and/or development of tools to support the IU such as code analyzers. The implementation phase covers the preparation of the learning environment, the training of the instructors and the application of the IU in the classroom.

An essential step in the instructional design process is the evaluation of the instructional unit in order to assess its quality and whether it allows the students to

Table 5
Common types of research design (Shadish, Cook, and Campbell, 2002) (Gresse von Wangenheim and Shull, 2009).

Type of study	Design	Representation X=Treatment O=Measurement R= Rrandom task
Case study	Only one post-test	X O
Case study	Only one post-test / pre-test	O X O
Quasi experimental	Static comparison group	X O O
	Static group pre-test / post-test	O X O O O
	Series of times	O O X O O
Experimental	Randomized post-test only	R X O R O
	Randomized pre-test / post-test	R O X O R O O
	Randomized with pre-test / post-test control group	R O X1 O R O X2 O

achieve the defined objectives (Branch, 2009). This evaluation is typically performed through an empirical study (Wohlin *et al.*, 2012). Different research designs can be adopted ranging from non-experimental studies (such as case studies) to experiments (Table 5).

In order to reach the evaluation objectives, the measurement must be explicitly defined in a way that draws a link between the objectives and the data collected and also generates a framework to analyze and interpret the data with the corresponding objectives (Wohlin *et al.* 2012). Several types of data collection instruments can be used, such as observation, questionnaire, interviews, or the artifacts created by the students themselves as well as test results (Branch, 2009). In case of online courses, data can also be collected in the form of log files. According to the objective of the evaluation and the nature of the data collected, different methods of qualitative or quantitative analysis can be used (Freedman *et al.*, 2007). The analyzed data are then interpreted, answering the analysis questions in order to achieve the evaluation goal.

4. Definition and Execution of the Systematic Mapping Study

To elicit the state of the art and practice on whether and how SE education is addressed in K-12, we conducted a systematic mapping study following the procedure proposed by Petersen *et al.* (2008).

4.1. *Definition of the Review Protocol*

The research question is: Are there (and what are their characteristics) instructional units that teach Software Engineering competences in the context of K-12? This research question is refined in the following analysis questions:

AQ1. Which IUs exist?
AQ2. Which SE competences are taught in the IUs?
AQ3. What are the instructional characteristics of the IUs?
AQ4. What are the context characteristics of the IUs?
AQ5. How were the IUs developed?
AQ6. How was the quality of the IUs evaluated?

Inclusion/exclusion criteria. We considered only peer-reviewed articles whose focus is to teach computation including SE competencies in K-12. Articles that focus on teaching computing in higher education and/or articles that present IUs for computer teaching without addressing SE concepts were excluded. We have also included secondary literature that has been discovered based on the references of the primary literature found (Verhoeff, 2006).

Quality Criteria. We considered only articles that present substantial information regarding the teaching of SE competencies, indicating, for example, lessons content, instructional material, etc.

Data source. We examined all published English-language articles that are available on Scopus (`www.scopus.com`) with free access through the CAPES Portal[1]. We also searched online education sites, including Udemy (`www.udemy.com`), Edx (`www.edx.org`), Khanacademy (`www.khanacademy.org`), Coursera (`www.coursera.org`) in order to discover instructional units taught as MOOCs.

Definition of the search string. The search string was composed of concepts related to the research question, including also synonyms, as indicated in Table 6.

From these keywords, the search string was calibrated and adapted according to the specific syntax of the data source as presented in Table 7. As a result of the calibration process, we also identified as relevant synonyms for the term "Software Enginerring" the terms UML and "software development process". The search of online courses was done via the MOOCs' sites limiting the category and subcategory to Information Technology and Software Development, respectively.

4.2. *Search Execution*

The search has been executed in March 2018 by the authors. The search has been done in two steps. In the first step the search was done via Scopus with the objective of finding articles about existing IUs. This search returned 466 articles (Table 8). From the search results, potentially relevant articles were selected according to the inclusion and exclusion criteria, quickly analyzing the title, abstract and keywords. As a result, 29 potentially relevant articles were identified. In the second selection stage, we analyzed the full text of the pre-selected articles to analyze their compliance with the inclusion/exclusion criteria and the quality criterion. As a result, 15 relevant articles were identified. All authors discussed the selection of papers until a consensus was reached.

Table 6
Keywords

Keyword	Synonyms
Software Engineering	UML, software development process
K-12	school
Teaching	learn, MOOC

Table 7
Search String

Source	Search String
Scopus	("software engineering" OR uml OR "software development process") AND (school OR "K-12") AND (teaching OR learn OR MOOC)

[1] A web portal for access to scientific knowledge worldwide, managed by the Brazilian Ministry on Education for authorized institutions, including universities, government agencies and private companies (`www.periodicos.capes.gov.br`).

Table 8
Amount of articles per selection stage

Source	Initial search results	Selected after 1° stage	Selected after 2° stage
Scopus	466	29	15

Several IUs found in the initial search were excluded, not presenting instructional units, such as for example IUs that only report the importance of SE education in K-12 (e.g., Bell *et al.*, 2014; Bollin *et al.*, 2016) or do not present details about SE teaching (e.g., Azenkot *et al.*, 2011).

In a second step, we also searched for MOOCs aiming at teaching SE on K-12 level. Relevant courses were selected using the same inclusion/exclusion criteria. As a result of this additional research one relevant IU was found (De Kereki and Manataki, 2015). Other courses for computing education in K-12 have been disregarded, if not explicitly covering SE concepts (e.g., course provided by the Technovation Challenge (Technovation, 2018)) and/or details of the IUs were not accessible (e.g., afsenyc.org).

Another IU was found by analyzing the references of the primary literature found in the searches (Verhoeff, 2006).

5. Data Analysis

To answer the analysis questions, we extracted relevant information from the encountered articles and course material as specified in Table 9.

The articles were read and the data were extracted by the authors. Extraction of the data was complicated in several cases by the way the studies were reported. As the publications in this area do not follow any structured protocol, the information to be extracted is not always presented explicitly. In these cases, information was inferred from the article, including for example, the description of the learning objectives, language of the IU, and the pre-requisites.

Table 9
Specification of the extracted information

Analysis question	Information extracted	Description
AQ1. Which IUs exist?	Reference	Bibliographic reference
AQ2. Which SE competences are taught in the IUs?	Learning objective(s) with respect to SE	Identification of the objective(s) describing what the learner should learn with respect to SE
	SE knowledge area(s)	SE knowledge areas addressed by the IU
	SE methods/technique(s)	SE methods/techniques addressed by the IU
	SE tool(s)	SE tools adopted for teaching in the IU

Continued on next page

Table 9 – continued from previous page

Analysis question	Information extracted	Description
AQ3. What are the instructional characteristics of the IUs?	General learning objective of the IU	Identification of the learning objective of the IU in general
	General description	Brief overview of the IU presenting its main characteristics
	Education mode	Identification of the education mode (in-class or distance/online)
	Programming environment(s)	Programming language/platforms used in the IU
	Instructional method	Instructional method(s) used in the IU
	Instructional material	Instructional material used in the IU
	Assessment method(s)/ instrument(s)	Method(s)/instrument(s) used for assessing students' learning in the IU
	Language(s)	Language(s) in which the IU is available
	License	Usage license of the IU
AQ4. What are the context characteristics of the IUs?	Educational stage	Educational stage for which the IU is designed
	Duration of the IU	Duration of the IU (number of hours/classes)
	Pre-requisites	Pre-requisites of students with respect to computing competencies
AQ5. How were the IUs developed?	Development method of the IU	Indication of the method used for the development of the IU
AQ6. How was the quality of the IUs evaluated?	Research design	Indication of the type of study (research design) adopted for the evaluation of the IU
	Factor(s) evaluated	Indication of the factors that were evaluated
	Data collection method(s)	Indication of the data collection method(s) adopted for the evaluation of the IU
	Sample size	Number of data points used for the evaluation
	Replicated studies	Indication of possible replications of the evaluation in various contexts
	Data analysis method(s)	Indication of the data analysis method(s) used for the evaluation of the IU
	Findings	Description of the main results, strengths and weaknesses of the IU identified

We also observed that the majority of the articles do not describe how the IUs were developed as well as lack information regarding their evaluation, for example, not addressing threats to validity. In case the article does not present any information to be extracted, we indicate the lack of this information as not informed (NI).

AQ1. Which IUs Exist?

As a result of the research, a total of 17 instructional units focused on computing education were identified that also approach the teaching of software engineering in K-12 (Table 10).

Table 10
Instructional Units

Reference	
(Bollin and Sabitzer, 2015)	Bollin, A; Sabitzer, B. (2015) Teaching Software Engineering in schools on the right time to introduce Software Engineering concepts. In: Proceedings of the Global Engineering Education Conference, Tallinn, Estônia, pp. 518–525.
(Collofello, 2002)	Collofello, J. S. (2002) Creation, deployment and evaluation of an innovative secondary school software development curriculum module. In: Proceedings of the 32nd Annual Frontiers in Education, Boston, MA, USA, pp. 1–4.
(Corbett and Nesiba, 2015)	Corbett, K.; Nesiba, N. (2015) Programming design process: Providing K-12 students with a structure to attain programming goals. In: Proceeding of the Frontiers in Education Conference, El Paso, TX, USA, pp. 1–4.
(De Kereki and Manataki, 2016)	De Kereki, I. F.; Manataki, A. Code Yourself! An introduction a programming. Available on: <https://pt.coursera.org/learn/intro-programming>. Acess: 05 Mar. 2018. De Kereki, I. F.; Manataki, A. (2016) "Code Yourself" and "A Programar": a bilingual MOOC for teaching Computer Science to teenagers. In: Proceeding of the Frontiers in Education Conference (FIE), Erie, PA, USA, pp. 1–9.
(Fronza *et al.*, 2017)	Fronza, I.; Ioini, N.; Corral L. (2017) Teaching Computational Thinking Using Agile Software Engineering Methods: A Framework for Middle Schools. ACM Transactions on Computing Education, v. 17, n. 4, pp 1–28.
(Fronza *et al.*, 2016)	Fronza, I.; Ioini, N.; Corral L. (2016) Blending Mobile Programming and Liberal Education in a Social-Economic High School. In: Porceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems, Austin, TX, USA, pp. 123–126
(Fronza *et al.*, 2015)	Fronza, I.; El Ioini, N.; 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, Berlin, Germany, pp. 21–26.
(Hermans and Aivaloglou, 2017)	Hermans, F; Aivaloglou, E. (2017) Teaching software engineering principles to K-12 students: a MOOC on Scratch. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track, Buenos Aires, Argentina, pp. 13–22.
(Köhler *et al.*, 2012)	Köhler, B; Gluchow, M; Brugge, B. (2012) Teaching Basic Software Engineering to Senior High School Students. In: Proceeding of the IADIS International Conference e-Society, Munich, Germany, pp. 149.
(Missiroli *et al.*, 2017)	Missiroli, M.; Russo, D.; Ciancarini, P. (2017) Agile for Millennials: A Comparative Study. In: Proceedings of IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM), Buenos Aires, Argentina, pp. 47–53
(Missiroli *et al.*, 2016)	Missiroli, M.; Russo, D.; Ciancarini, P. (2016) Learning Agile software development in high school: an investigation. In: Proceedings of the 38th International Conference on Software, Austin, TX, USA, pp. 293–302,
(Rusu, A *et al.*, 2011)	Rusu, A. *et al.* (2011) Employing software maintenance techniques via a tower-defense serious computer game. In: Proceedings of the International Conference on Technologies for E-Learning and Digital Entertainment, Berlin, Germany, pp. 176–184.
(Rusu *et al.*, 2010)	Rusu, A. *et al.* (2010) Learning software engineering basic concepts using a five-phase game. In: Proceedings of IEEE Frontiers in Education Conference, Washington, DC, USA, pp. 1–6.
(Sarkar and Bell, 2013)	Sarkar, A; Bell, T. (2013) Teaching black-box testing to high school students. In: Proceedings of the 8th Workshop in Primary and Secondary Computing Education, Aarhus, Denmark, pp. 75–78.

Continued on next page

Table 10 – continued from previous page

Reference	
(Serrano and Serrano, 2013)	Serrano, M.; Serrano, M. (2013) Requisitos ao Código: Uma Proposta para o Ensino da Engenharia de Software no Ensino Médio. In: Proceedings of the International Requirements Engineering Conference, Rio de Janeiro, Brazil, pp. 37–42.
(Starrett, 2007)	Starrett, C. (2007) Teaching UML Modeling Before Programming at the High School Level. In: Proceedings of Seventh IEEE International Conference on Advanced Learning Technologies, Niigata, Japan, pp. 713–714.
(Verhoeff, 2006)	Verhoeff, T. (2006) A master class software engineering for secondary education. In: Proceeding of the International Conference on Informatics in Secondary Schools-Evolution and Perspectives. Heidelberg, Germany, pp. 150–158.

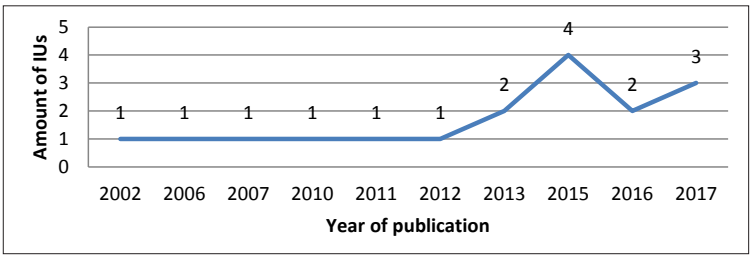

Fig. 1. Amount of IUs focusing on SE education in K-12 published per year.

This shows that so far very few IUs approach SE education in K-12, yet with a slight increase since 2013, probably also related to the trend of increasing computing education in K-12 in general (Fig. 1).

AQ2. Which SE competences are taught in the IUs?

The IUs teach competencies related to several SE knowledge areas in accordance to the SWEBOK (IEEE CS, 2014). Among the areas most frequently approached by the IUs are the areas related to the main phases of the software process: software requirements, software engineering models and methods, software construction and software testing (Fig. 2). The knowledge area most taught by these IUs is software testing including unit testing, functional testing, and acceptance testing. The development of software requirements is also widely taught by applying various requirements gathering and analysis techniques, such as user stories, use case diagrams, storyboards, software requirements specifications, among others. Software engineering models and methods are also covered by several IUs, using flow diagrams, pseudocode, UML class diagrams and state machines to visualize the architecture and algorithms of software systems. IUs focusing on mobile application development also address interface design by creating paper prototypes of the screens. Regarding software construction, techniques such as the creation of understandable source code (naming), code reuse and pair programming are addressed. A detailed summary of the data extracted with respect to the SE knowledge areas covered by each of the IUs is presented in Appendix A.

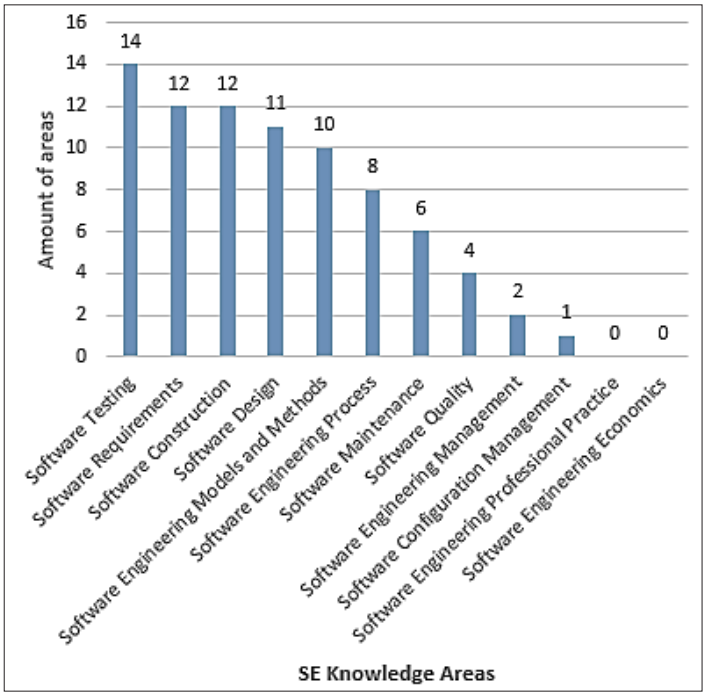

Fig. 2. Freqeuncy of SE Knowledge Areas covered by the IUs.

A third of the IUs stands out by addressing topics related software maintenance. These IUs include the teaching of competencies related to the re-engineering, adaptation and/or evolution of pre-existing software. What apparently facilitates the teaching of maintenance concepts in the context of K-12 are block-based programming environments, such as Scratch, which strongly stimulate and support the remix of programs (Brennan and Resnick, 2013). In this context, Rusu *et al.* (2011) adopt an educational game that teaches the four types of software maintenance as an alternative instructional strategy.

Several IUs also explicitly teach concepts related to the software process, life cycle models, and development methodologies, which are typically in an implicitly and simplified manner covered in instructional units teaching programming. We observed that mainly simple life-cycle models such as the waterfall and iterative models are addressed and/or agile methodologies such as Scrum, Extreme programming (XP), Model-Driven Development (MDD) and Test-Driven Development (TDD) (Fig. 3). Alternatively a general engineering process, the Programming Design Process (PDP) is presented.

Basically, all IUs are intended to lead the student learning SE competencies at the application level and are designed to give students the opportunity to execute SE processes (Appendix A). This includes IUs that aim to execute all the main phases of the software process as well as others that focus only on specific phases of the process, taking into account practical restrictions regarding the duration of the IU. Several IUs

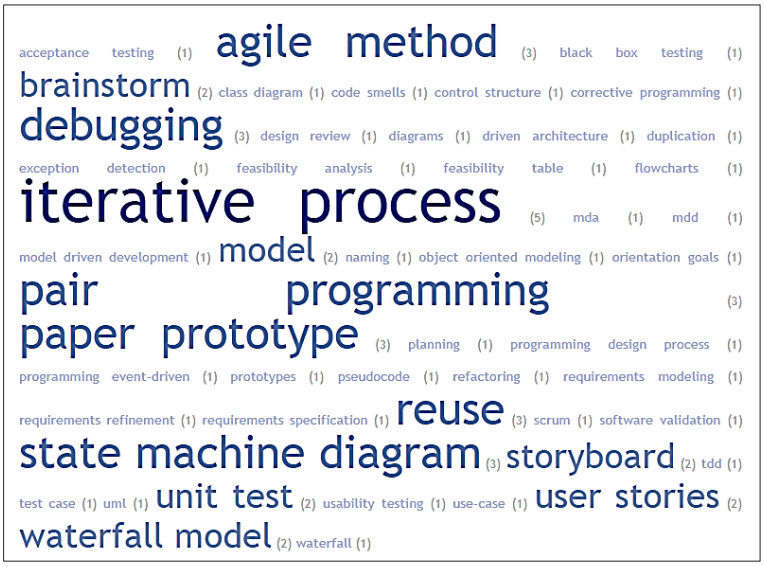

Fig. 3. Frequency of the models/methods/techniques addressed in SE education in K-12.

are applied in a multidisciplinary way integrated in other disciplines of the K-12 curriculum, such as Physics and Arts. Very few IUs use SE tools to aid teaching and/or to teach their use. Only two IUs report the use of case tools (Rational Rose and NetBeans). Other tools that have been adopted include testing tools (JUnit, a test case validation tool (testBedv9.html)).

AQ3. What are the instructional characteristics of the IUs?

The teaching of SE competencies is typically integrated in IUs generally focusing on teaching algorithms and programming aimed at the development of animations, web/mobile applications, or robotics. Therefore, generally block-based visual programming environments are used for novices, such as Scratch, Alice and App Inventor or text-based programming languages, such as Java, Delphi Pascal, ANSI C and R language for statistical computation (Fig. 4). To teach the development of robotic programs the IUs used PBASIC, a microcontroller-based version of BASIC or the Lego Robot programming language.

Few IUs are specifically focused on teaching SE competencies, such as the Tower-Defense Serious Computer Game (Rusu *et al.*, 2011), to teach software maintenance concepts. Another example is the IU presented by Sarkar and Bell (2013) aiming at the creation of test cases for pre-existing software.

With respect to the instructional methods, there is a strong predominance of active learning approaches aiming at the achievement of learning objectives on the application level. Several IUs involve software development assignments. These range from tasks with a well-defined specification of the problem to be solved for which an expected solution exists to tasks with ill-defined problems without a previously known solution.

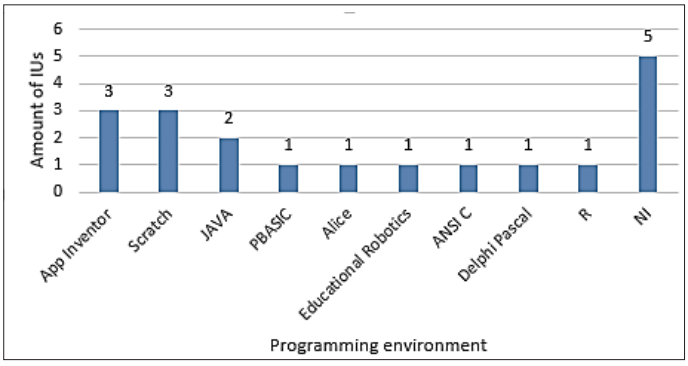

Fig. 4. Programming environments used in SE education in K-12.

Although focusing more on active learning, several IUs also include other direct instructional methods as lectures, especially in the initial part of the IU (Fig. 5). Interactive methods such as co-operative learning, challenges and discussions were also used. Two IUs have adopted game-based learning (Rusu *et al.*, 2011; Rusu *et al.*, 2010).

According to this variety of instructional methods, several types of instructional material are used (Fig. 6). The material most commonly used is software artifacts. These artifacts are typically used to assist in the application of SE teaching, including e.g., specification of requirements of a predefined software system, use cases, user stories, code samples, among others. Some IUs also used a complete software system (i) to teach the creation of acceptance tests (Sarkar and Bell, 2013), (ii) to exemplify a solution and/or (iii) as a teaching strategy (Rusu *et al.*, 2010). Instructional videos, tutorials, forums, etc. are specific to IUs designed as online courses. Several IUs also use exercise sheets, workbooks or journals to record the students' experiences. However, in general, we observed a lack of information regarding the instructional material, their availability

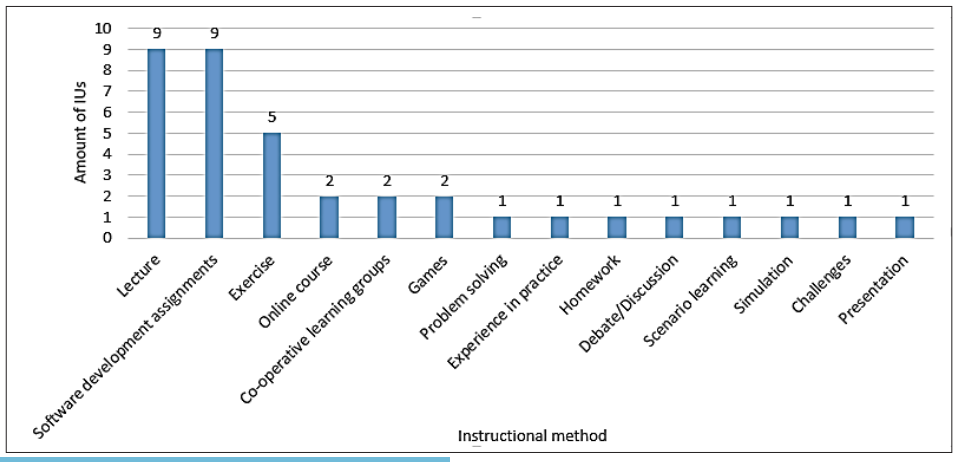

Fig. 5. Instructional Methods used for SE education in K-12.

and license, which makes it difficult for others to use them. Most materials are available in one language only (predominantly in English), which may also limit a broader adoption of IU in other countries that typically require instructional material in the native language at this educational stage.

Student learning is assessed primarily through performance-based assessments analyzing artifacts created in the context of the software process and/or software programs either by the instructor or through peer reviews. In some cases artifact-based interviews are used. Several IUs also adopt tests, quizzes or use the game score for the students' assessment (Fig. 7).

An overview of the information extracted with respect to the instructional characteristics of the IUs is presented in Appendix B.

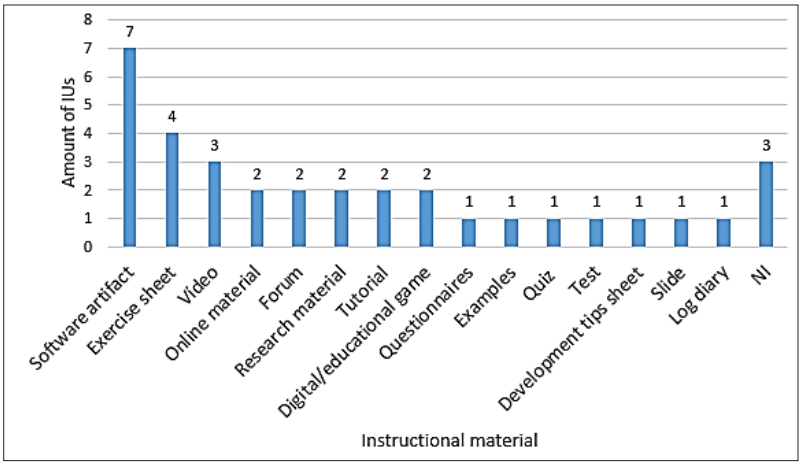

Fig. 6. Types of instructional material used.

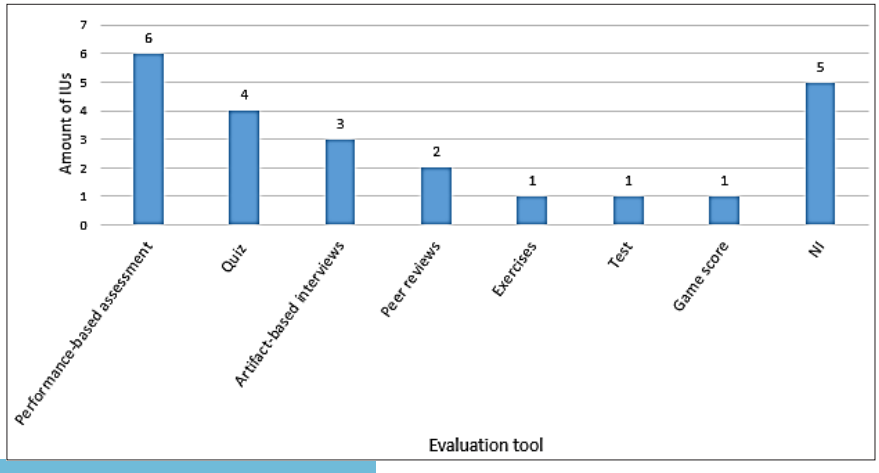

Fig. 7. Types of assessment methods used.

AQ4. What are the context characteristics of the IUs?

Most of the IUs are focused on teaching SE in high school (Fig. 8). No specific IU teaching in elementary school was found. Observing this predominance of IUs found for high school, the question that remains is why there are almost no IUs in elementary school. Yet, several authors report that the insertion of SE education can be beneficial even in elementary school, obviously in a limited way taking into account students' previous knowledge and curricular restrictions (Bollin and Sabitzer, 2015).

The encountered articles do not address the specific type of school (e.g., secondary technical school present in some countries), indicating that the presented instructional units have been applied in schools with a general focus.

The duration of the IUs varies largely from short and focused activities (30 minutes) to long-term courses (one year). According to the students' current lack of knowledge regarding computing and/or SE, most IUs are aimed at beginners with no prior computing/SE competencies. Only four IUs indicate the need for prior competencies mainly in relation to programming (Table 11).

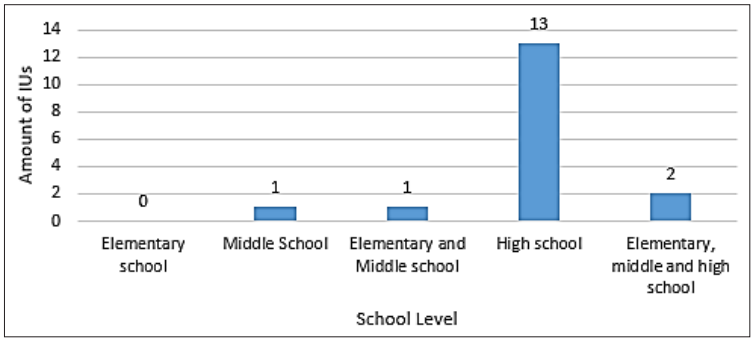

Fig. 8. Educational stages.

Table 11
Overview of IU context characteristics

Reference	Educational stage	Duration of the IU	Pre-requisites
(Bollin and Sabitzer, 2015)	High school	NI	none
(Collofello, 2002)	High school	NI	NI
(Corbett and Nesiba, 2015)	High school	NI	NI
(De Kereki and Manataki, 2016)	Elementary, middle and high school	15–20 hours	none
(Fronza *et al.*, 2017)	Middle school	60h (4h per week)	NI

Continued on next page

Table 11 – continued from previous page

Reference	Educational stage	Duration of the IU	Pre-requisites
(Fronza *et al.*, 2016)	High school	20–30 hours	none
(Fronza *et al.*, 2015)	High school	5 days (with a total of 40 hours)	none
(Hermans and Aivaloglou, 2017)	Elementary and middle school	6 weeks with a total of 12 to 36 hours (estimating a weekly effort of 2 to 6 hours)	none
(Köhler *et al.*, 2012)	High school	3 days	none
(Missiroli *et al.*, 2017)	High school	25 hours	Minimum of 2 years programming experience
(Missiroli *et al.*, 2016)	High school	case 1: 2 hours, case 2: 5.5 hours	1 year experience in programming in Java OO/html, web service, databases
(Rusu *et al.*, 2011)	Elementary, middle and high school	NI	Prior programming competence
(Rusu *et al.*, 2010)	High school	NI	none
(Sarkar and Bell, 2013)	High school	30 minutes	NI
(Serrano and Serrano, 2013)	High school	10 hours and 5 class	NI
(Starrett, 2007)	High school	1 year	NI
(Verhoeff, 2006)	High school	3 days	Prior competence regarding programming (global and local variables and procedures)

AQ5. How were the IUs developed?

To achieve effective learning outcomes, IUs need to be developed systematically following instructional design models. However, we observed a general lack of information in the articles in relation to the way the IUs were developed (Table 12). Few publications provide information on this issue, typically only by indicating the stakeholders involved in the IU development, through cooperations between schools and/or universities involving teachers, instructors and tutors. Only Köhler *et al.* (2012) explicitly reports the instructional principles used in the IU development (goal-based scenarios (Schank, 1996; Schank, 1992; Schank *et al.*, 1994) and scaffolding (Vygotsky, 1978)).

This lack of information provided on how the IUs were developed, clearly points out a need for a stronger adoption of systematic methods for the development of such instructional units, following not only all required phases of instructional design, but also the applying well-accepted and sound models, methods and techniques.

AQ6. How was the quality of the IUs evaluated?

An essential as part of the systematic development and improvement of an IU is the evaluation of the IU. Evaluation is typically performed through empirical studies in

Table 12
Methods used to develop the IUs

Reference	Development method of the IU
(Bollin and Sabitzer, 2015)	NI
(Collofello, 2002)	IU developed in cooperation with of local school teachers. Graduate students participated in the construction of instructional material and in class.
(Corbett and Nesiba, 2015)	NI
(De Kereki and Manataki, 2016)	The IU was developed by an international and multidisciplinary team. The design and creation process was carried out in cooperation taking into account the target audience. Learning levels were defined based on Bloom's taxonomy. Course material has been developed in detail. The course was pretested with different groups in 2 countries.
(Fronza *et al.*, 2017)	NI
(Fronza *et al.*, 2016)	NI
(Fronza *et al.*, 2015)	NI
(Hermans and Aivaloglou, 2017)	The course was developed as a MOOC (Open Massive Online Course) on the edx platform (https://www.edx.org/). Thus, the IU has been designed following the pattern of edx platform courses, consisting of videos, quizzes and forum interactions.
(Köhler *et al.*, 2012)	The IU design adopts the principle of goal-based scenarios (Schank 1996; Schank 1992; Schank *et al.*, 1994) and the scaffolding principle (Vygotsky 1978). The IU was created through a case study including the following steps: analysis of the target group; brainstorming of possible topics of interest; topic selection; creation of object model; project development, estimation of effort for students; create project "scaffold"; coding tutorial; template; distribution of tasks in teams.
(Missiroli *et al.*, 2017)	NI
(Missiroli *et al.*, 2016)	NI
(Rusu *et al.*, 2011)	NI
(Rusu *et al.*, 2010)	The games were developed by teams involving graduate and undergraduate students.
(Sarkar and Bell, 2013)	NI
(Serrano and Serrano, 2013)	NI
(Starrett, 2007)	NI
(Verhoeff, 2006)	NI

the classroom. Several IUs were evaluated by means of a case study (Fig. 9). In these studies, the evaluation was systematically defined and, during and after the treatment (teaching SE), data was collected in relation to the objective of the evaluation. Only two studies adopted a more rigorous research design. Missiroli *et al.* (2017) conducted an experiment comparing the performance and satisfaction of students and teachers with respect to the use of two software development approaches in computing education, the waterfall model and Scrum. Fronza *et al.* (2017) adopt a quasi-experimental approach to evaluate the effectiveness of a framework based on agile SE methods to teach com-

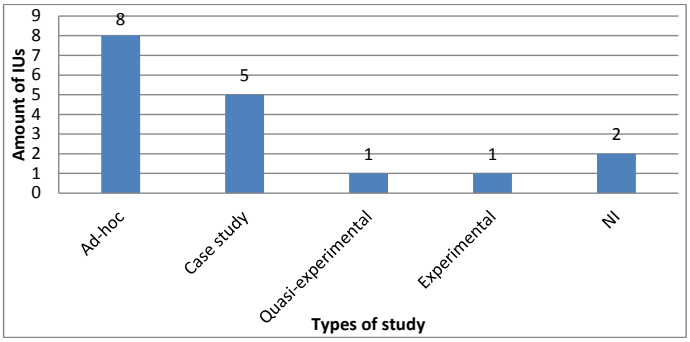

Fig. 9. Types of studies adopted for the evaluation of the IUs.

puting in high school. Both experimental studies follow the methodology proposed by Wohlin *et al.* (2012). A considerable number (8 IUs) were evaluated in a less rigorous way through ad-hoc evaluations, without detailed definition of the evaluation objectives, measurement and data analysis. As a result, these studies typically only comment on students' informal feedback and/or observations during the application.

Most studies evaluate more than one quality factor (Fig. 10). Learning is the most evalued quality factor. This shows that, in fact, the main concern is the learning effect provided by the IUs. The assessment of this factor usually refers to improving competence by comparing the level of competence of students after the IU with their level of competence before the IU, usually based on a pre/post-test score. None of the studies evaluate the learning effect in relation to the different learning levels, for example, based on Bloom's taxonomy. The IU's efficiency is evaluated based mainly on the analysis of the students' learning and the feedback received by the students. Several studies also assess the degree of satisfaction to evaluate whether students feel that their dedicated effort results in learning. Comparing the factors being evaluated, we can observe a lack of conformity between the studies. With the exception of the evaluation of the degree of

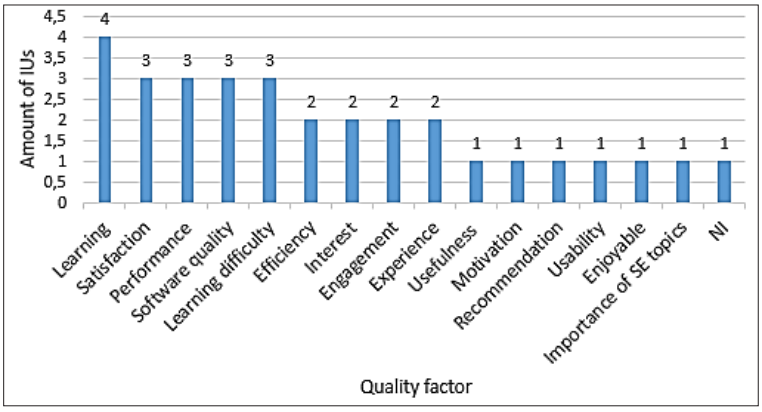

Fig. 10. Quality factors being evaluated in the studies.

students' learning evaluated in most studies, the factors analyzed vary greatly indicating the lack of a commonly accepted evaluation model for this type of IU. Besides evaluating the impact of the IUs several evaluations also included the measurement of feedback on the IU itself and/or the programming environment, as well the qualitative indication of strengths and weaknesses observed.

Data regarding the evaluation is collected in several ways (Fig. 11). Most of the data is collected via questionnaires after IU application. Several studies also extract data based on the performance-based assessment of artifacts created by students during the IU, exercises or tests. In some cases teachers and/or students are interviewed at the end of the IU in order to obtain information on the learning experience, learning environment, and motivation. Observations were typically used to analyze student performance, but also to evaluate their enjoyment and satisfaction. Log files from forums, videos, wikis, etc. were mostly used by online courses to analyze the students' engagement in the course.

Taking into consideration the less rigorous research designs adopted, most studies only perform qualitative data analyses and/or quantitative analyses in a descriptive way (Fig. 12). Only two studies report the usage of statistical tests (Hermans and Aivaloglou, 2017; Missiroli *et al.*, 2017).

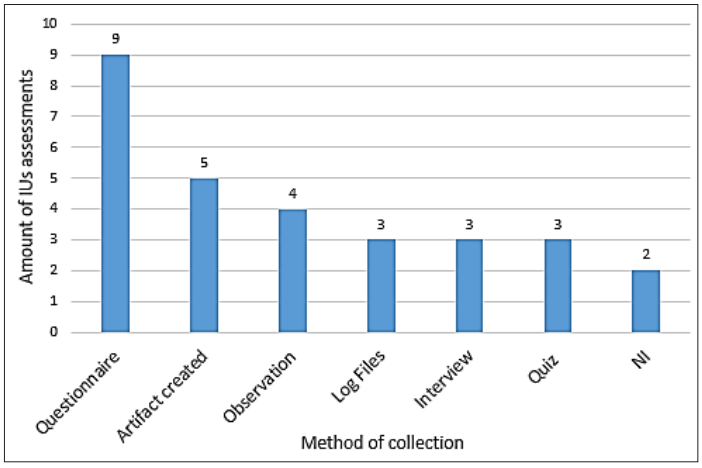

Fig. 11. Data collection methods used for the evaluation of the IUs.

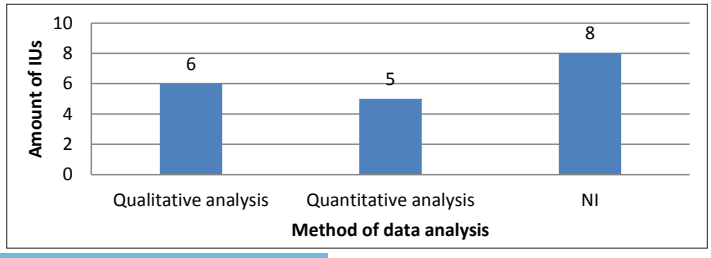

Fig. 12. Usage of data analysis methods.

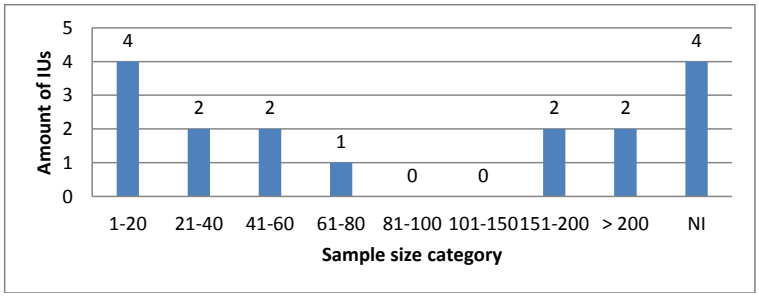

Fig. 13. Amount of IU evaluations by sample size.

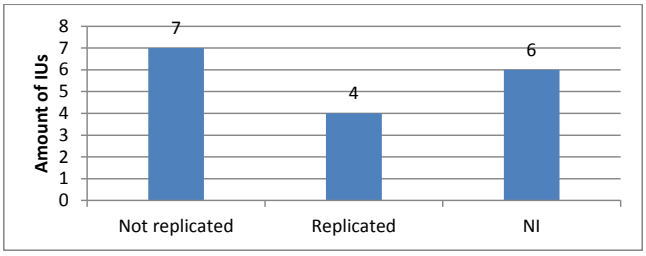

Fig. 14. Amount of replicated studies.

As shown in Fig. 13, most evaluations were performed with small samples, ranging from 1 to 60 participants. This low number of participants usually corresponds to the size of a class in which the IU is applied and evaluated. However, four evaluations were performed with more than 150 participants. Some studies (4 IUs) did not report the sample size.

Almost half of the studies were replicated in more than one context, contributing to the external validity of the findings (Fig. 14). However, a large part of the replications still occurred only through a single study in a specific context, usually by the IU creators themselves.

An overview on the information extracted with respect to the evaluation of the IUs is presented in Appendix C.

6. Discussion

Considering the importance of SE in software development, a very small number of instructional units (only 17 in total) were found, aiming to teach these important competences in K-12. These IUs focus on the main phases of the software process, including software requirements, software engineering models and methods, software construction and software testing. Some IUs also explicitly address software maintenance. Some of the IUs follow a traditional life-cycle model, such as the waterfall model or the V model. On the other hand, several IUs adopt agile methodologies following an iterative process

and creating artifacts such as user stories, storyboards, etc. This indicates that both the adoption of simple and/or iterative life cycle models may be beneficial to introduce the software process at this educational stage.

SE education in K-12 is mostly concentrated in high school, although some authors also report observed benefits from the introduction of teaching SE in elementary school (Bollin and Sabitzer, 2015). Despite this, none of the authors reports difficulties or low performance of students in learning SE concepts. In addition, Bollin and Sabitzer (2015) conclude that teaching of SE can be started in elementary school without difficulties. However, it is necessary to identify which knowledge areas, content and level of detail should be taught according to the student's age (Bollin and Sabitzer, 2015). Another finding concludes that K-12 students demonstrate no difference in performance in learning SE and programming (Hermans and Aivaloglou, 2017).

Few IUs are focused exclusively on teaching SE, most IUs teach SE competencies while teaching programming. Bollin and Sabitzer (2015) conclude that there is no need for students to have previous computing experience and that SE can be taught with basics of programming logic and modeling (flow diagrams). An exception is the IU presented by Starrett (2007) that teaches software design using the UML notation before teaching programming. In accordance to the author, modeling and abstraction are fundamental to analytical thinking. He also points out that modeling provides a method to help students address problems and solutions step-by-step. The results of this study show that students learned the core concepts of abstraction in a quick and natural way.

The IUs either focus on teaching several phases of the software process or focus only on a specific process by pre-defining the input artifacts for this phase. This may represent a teaching alternative, especially when there are time constraints on the IU. The majority of the IUs is integrated in the context of IUs focused on programming teaching, few explicitly focus on teaching SE concepts. The integration of SE teaching into IUs teaching programming can be beneficial both in relation to time constraints and in learning a broader and more diverse understanding of the area of computation. In order to enable the adoption of computing/SE education, several IUs are carried out in a multidisciplinary way integrated in other disciplines of the K-12 curriculum such as Physics or Arts.

In general, on this educational stage basic SE concepts related to the cognitive domain are addressed with a scope varying in relation to the duration of the IU. Weaknesses typically observed in relation to SE education in higher education restricted to small-scale projects lacking real project characteristics are even more present in IUs found in K-12 (Malik *et al.*, 2012). IUs in K-12 are even more focused on teaching basic SE knowledge, not addressing more comprehensive skills and practical experiences.

We also observed a preference for the adoption of agile methodologies that seem more appropriate to initiate the teaching of SE. Interestingly, only seven IUs report the use of CASE tools, as the use of this type of tool helps to carry out the activities of higher quality processes (IEEE CS, 2014). Thus, the question that arises is if the teaching of CASE tools is inappropriate on this educational stage and/or if it is caused by the lack of this type of functionality in the programming environments typically used in K-12.

The majority of IUs aims at teaching SE competencies at the level of application by adopting active learning approaches. Typically, after an introductory part, students develop software (animations, web/mobile applications or robots). The assessment of the students' learning is usually based on their performance analyzing the artifacts created by the students and/or quizzes. However, no detailed information on the assessment criteria have been reported, such as for example rubrics, etc.

Aiming at disseminating the IUs presented in the articles, we observed, in general a lack of availability of detailed information and/or instructional material. The vast majority of IUs have been created in only one language and are not accessible, neither free nor paid. This unavailability of IUs may hinder a larger scale application. Another issue we observed is the lack of support for the training of instructors in order to prepare them adequately for the application of the IUs in the classroom. Taking into account that today there is a lack of K-12 teachers with computing background, leaving as a solution the adoption a multidisciplinary approach in which computation is taught by teachers trained in other disciplines. Therefore, motivation and in-service teacher training become crucial, since they need to have computing, SE and technological knowledge as well as knowledge of relevant pedagogical content (Gal-Ezer and Stephenson, 2010; Bollin and Sabitzer, 2015).

Another issue is the fact that many articles do not present essential information regarding the learning objective(s) and/or instructional strategy, nor do they indicate the methodology used to develop the IUs. This weakness can also be observed in relation to the evaluation of most IUs. Several do not report evaluations or performed them only in an ad-hoc manner, which leaves the reported results questionable. The large variation of the factors evaluated in different ways also indicates the lack of evaluation models in this area to facilitate a more uniform evaluation of these IUs.

6.1. *Threats to Validity*

As in any systematic mapping studies, some threats to validity of the results exist. We, therefore, identified potential threats and applied mitigation strategies in order to minimize their impact.

Publication bias. Systematic mappings may suffer from the common bias that positive outcomes are more likely to be published than negative ones. However, we consider that the findings of the articles, whether positive or negative, have only a minor influence on this systematic mapping since we sought to characterize the approaches rather than analyze their impacts on learning.

Identification of studies. Another risk is the omission of relevant studies. In order to mitigate this risk, we carefully constructed the search string to be as inclusive as possible, considering not only core concepts but also synonyms. We also searched prominent online course bases in order to reduce the risk of excluding existing IUs that have not yet been reported through scientific articles. Furthermore, we also included secondary literature identified based on the references of the primary literature identified in the search.

Selection and extraction of study data. Threats to study data selection and extraction were mitigated by providing a detailed definition of inclusion/exclusion and qual-

ity criteria. We defined and documented a rigid protocol for the study selection and all authors performed the selection together, discussing the selection until consensus was reached. Data extraction was hindered in some cases, as the relevant information was not always presented explicitly and/or using a commonly accepted terminology and, therefore, in some cases had to be inferred. However, this inference was made by the first author and carefully reviewed by the co-authors.

7. Conclusion

In this article, we present the state of the art and practice on teaching SE competences in K-12. We have identified only 17 IUs mainly focused on high school. The IUs mostly address the main stages of the software process, including software requirements, software engineering models and methods, software construction and software testing, typically adopting either traditional life cycle models or agile methodologies. In general the teaching of SE is inserted in IUs mainly aimed at teaching programming, often also in a multidisciplinary way integrated into other disciplines of the K-12 curriculum, such as Physics or Arts. The majority of IU aims at learning SE competencies at the level of application by adopting active learning approaches leading the student to create artifacts related to the software process.

Even with the authors reporting the benefits observed and the success of SE education in K-12, we observed that these results may be questionable taking into account the limited information on how IUs were developed and the low scientific rigor in their evaluation with often small samples. We also note that the vast majority of IUs have been created in only one language and are not accessible, neither free nor paid, which may hinder their application on a larger scale. Based on the results of our review, it becomes obvious that there is a need for the development of IUs focused on SE education in K-12 popularizing not only programming but also SE competence as an essential area of computing.

Acknowledgment

This work is supported by the CNPq (National Council for Scientific and Technological Development), a Brazilian government entity focused on scientific and technological development.

References

ACM/IEEE (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*.
www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

ACM/IEEE (2014). *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*.
https://www.acm.org/binaries/content/assets/education/se2014.pdf

Alice. *Alice – Tell Stories. Build Games. Learn to Program*. https://www.alice.org/
App Inventor. *App Inventor – Explore MIT App Inventor*. http://appinventor.mit.edu/explore/
Azenkot, S. *et al.* (2011) Overcoming barriers among Israeli and Palestinian students via computer science. *In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 667–672.
Bascou, N. A.; Menekse, M. (2016) Robotics in K-12 formal and informal learning environments: A review of literature. *In: Proceedings of the 2016 ASEE Annual Conference & Exposition*, 1–46.
Blackgirlscode. *Black Girls Code imagine. build. create. – Black Girls Code, BlackGirlsCode, Women of Color in Technology*. http://www.blackgirlscode.com
Blockly. *Block Google Developers*. https://developers.google.com/blockly/
Bloom, B.,S. (1956). *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*. Boston: Addison-Wesley Longman Ltd.
Bollin, A. *et al.* (2016) Software engineering in primary and secondary schools-Informatics education is more than programming. *In: Proceedings of the 29th International Conference on Software Engineering Education and Training*, 132–136.
Bollin, A; Sabitzer, B. (2015) Teaching Software Engineering in schools on the right time to introduce Software Engineering concepts. *In: Proceedings of the Global Engineering Education Conference*, 518–525.
Brennan, K; Resnick, M. (2013) Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. *In: Proceedings of the Conference on Emerging Technologies for the Classroom,* 253–268.
Code. *Anybody can learn* | Code.org. https://code.org/
Codeclubworld. *Code Club International – A worldwide network of coding clubs for children*. https://www.codeclubworld.org/
Collofello, J.S. (2002) Creation, deployment and evaluation of an innovative secondary school software development curriculum module. *In: Proceedings of the 32nd Annual Frontiers in Education*, 1–4.
CNE. *Iniciativa Computação na Escola*. http://www.computacaonaescola.ufsc.br/
Corbett, K.; Nesiba, N. (2015) Programming design process: Providing K-12 students with a structure to attain programming goals. *In: Proceedings of the Frontiers in Education Conference*, 1–4.
CSTA (2017). *K-12 Computer Science Framework*. http://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf
De Kereki, I.F.; Manataki, A. (2016) "Code Yourself" and "A Programar": a bilingual MOOC for teaching Computer Science to teenagers. *In: Proceedings of the Frontiers in Education Conference (FIE)*, 1–9.
De Kereki, I.F.; Manataki, A. *Code Yourself! An introduction a programming.* https://pt.coursera.org/learn/intro-programming
Freedman, D. *et al.* (2007). *Statistics*. New York: WW Norton.
Fronza, I. *et.al*. (2017) Teaching Computational Thinking Using Agile Software Engineering Methods: A Framework for Middle Schools. *ACM Transactions on Computing Education*, 17(4), 1–28.
Fronza, I. *et al.* (2016) Blending Mobile Programming and Liberal Education in a Social-Economic High School. *In: Proceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems*, 123–126.
Fronza, I. *et al.* (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*, 21–26.
Girlswhocode. *Girls Who Code – Join 90,000 Girls Who Code today!*. https://girlswhocode.com
Gresse von Wangenheim, C. *et al.* (2018) CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs. *Informatics in Education*, 17(1), 117–150.
Gresse von Wangenheim, C.G.; Shull, F. (2009) To Game or not to Game?. *IEEE Software*, 26(2), 92–94.
Grover, S.; Pea, R. (2013) Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43.
Grover, S. *et al.* (2015). Designing for deeper learning in a blended computer science course for middle school students. *Journal of Computer Science Education*, 25(2), 199–237.
Heintz, F. *et al.* (2016) A review of models for introducing computational thinking, computer science and computing in K-12 education. *In: Proceedings of the Frontiers in Education Conference (FIE)*, 1–9.
Heredia, A. *et al.* (2015) A Systematic Mapping Study on Software Process Education. *In: Proceedings of the International Workshop on Software Process Education, Training and Professionalism*. 7–17.
Hermans, F; Aivaloglou, E. (2017) Teaching software engineering principles to K-12 students: a MOOC on Scratch. *In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, 13–22.
Hubwieser, P. *et al.* (2015) A global snapshot of computer science education in K-12 schools. *In: Proceedings*

of the 2015 ITiCSE on Working Group Reports, 65–83.
Hubwieser, P. (2012) Computer Science Education in Secondary Schools –The Introduction of a New Compulsory Subject. *ACM Transactions on Computing Education*, 12(4), 16.
IEEE. (2010). *ISO/IEC/IEEE 24765:2010 Systems and Software Engineering —Vocabulary*.
IEEE CS. (2014). *SWEBOK – Guide to the Software Engineering body of knowledge* (3th ed.). Silver Spring/USA: IEEE
Gal-Ezer, J.; Stephenson, C. (2010) Computer science teacher preparation is critical. *ACM Inroads*, 1(1), 61–66.
Köhler, B *et al.* (2012) Teaching Basic Software Engineering to Senior High School Students. *In: Proceedings of the IADIS International Conference e-Society*, 149.
Kosa, M. *et al.* (2016) Software engineering education and games: a systematic literature review. *Journal of Universal Computer Science*, 22(12), 1558–1574.
Lye, S.Y.; Koh, J.H.L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, 41(C), 51–61.
Malik, B. *et al.* (2012) A systematic mapping study on software engineering education. *World Academy of Science, Engineering and Technology*, 6(11), 1974–1984.
Maiorana, F. *et al.* (2015). Quizly: A live coding assessment platform for App Inventor. *In: Proceedings of IEEE Blocks and Beyond Workshop*, 25–30.
Mead, N.R. (2009) Software engineering education: How far we've come and how far we have to go. *Journal of Systems and Software*, 82(4), 571–575.
Missiroli, M. *et al.* (2017) Agile for Millennials: A Comparative Study. *In: Proceedings of IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)*, 47–53.
Missiroli, M. *et al.* (2016) Learning Agile software development in high school: an investigation. *In: Proceedings of the 38th International Conference on Software*, 293–302.
MITa. *MIT App inventor – Explore*. `http://appinventor.mit.edu/explore/`
MITb. *Scratch*. `https://Scratch.mit.edu/`
Moreno-León, J.; Robles, G. (2016) Code to learn with Scratch? A systematic literature review. *In: Proceedings of the Global Engineering Education Conference (EDUCON)*, 150–156.
Petersen, K. *et al.* (2008) Systematic mapping studies in software engineering. *In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, 68–77.
Pokress, C.; Veiga, J.D. (2013). MIT App inventor: Enabling personal mobile computing. *In: Proceedings of Programming for Mobile and Touch*, 3.
Rusu, A. *et al.* (2011) Employing software maintenance techniques via a tower-defense serious computer game. *In: Proceedings of the International Conference on Technologies for E-Learning and Digital Entertainment*, 176–184.
Rusu, A. *et al.* (2010) Learning software engineering basic concepts using a five-phase game. *In: Proceedings of IEEE Frontiers in Education Conference*, 1–6.
Sarkar, A; Bell, T. (2013) Teaching black-box testing to high school students. *In: Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, 75–78.
Saskatchewan Education. (1991) *Instructional Approaches: A Framework for Professional Practice*. Saskatchewan Education, Canada.
Schank, R.C. (1992) *Goal-Based Scenarios. Technical Report #36.* Institute for the Learning Sciences, Northwestern University.
Schank, R.C. *et al.* (1994) The design of goal-based scenarios. *Journal of the Learning Sciences*, 3(4), 305–345.
Schank, R.C. (1996) *Goal-based scenarios: CASE-based reasoning meets learning by doing, in CASE-based reasoning: Experiences, lessons & future directions*, *D.Leake (ed.)*. AAAI Press/The MIT Press.
Scratch. *Scratch – Imagine, Program, Share*. `https://scratch.mit.edu/`
Serrano, M.; Serrano, M. (2013) Requisitos ao Código: Uma Proposta para o Ensino da Engenharia de Software no Ensino Médio. *In: Proceedings of the International Requirements Engineering Conference*, 37–42.
Shackelford, R. *et al.* (2005) *Computing curricula: The overview report*. `https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2005-march06final.pdf`
Shadish, W.R. *et al.* (2002). *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin Company, New York.
Shaw, M. (2000) Software engineering education: a roadmap. *In: Proceedings of the Conference on the Future of Software Engineering.* 371–380.
Starrett, C. (2007) Teaching UML Modeling Before Programming at the High School Level. *In: Proceedings of 7th IEEE International Conference on Advanced Learning Technologies*, 713–714.

Technovation. (2018) *Technovation Challenge*. https://technovationchallenge.org
US Departament Education (2018). *Organization of U.S. Education*. https://www2.ed.gov/about/offices/list/ous/international/usnei/us/edlite-org-us.html
Verhoeff, T. (2006) A master class software engineering for secondary education. *In: Proceedings of the International Conference on Informatics in Secondary Schools-Evolution and Perspectives*, 150–158.
Vygotsky, L.S. (1978) Mind in society: The development of higher mental processes. *Cambridge: Harvard University Press*, 159.
Wing, J.M. (2008) Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London: mathematical, physical and engineering sciences*, 366(1881), 3717–3725.
Wohlin, C. *et al.* (2012) *Experimentation in Software Engineering*. Springer-Verlag, Berlin.

F. da Cruz Pinheiro, is a master student of the Graduate Program in Computer Science (PPGCC) at the Federal University of Santa Catarina (UFSC) and a research student at the initiative Computing at Schools/INCoD/INE/UFSC.

C.G. von Wangenheim, is a professor at the Department of Informatics and Statistics (INE) of the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil, where she coordinates the Software Quality Group (GQS) focusing on scientific research, development and transfer of software engineering models, methods and tools and software engineering education in order to support the improvement of software quality and productivity. She also coordinates the initiative Computing at Schools, which aims at bringing computing education to schools in Brazil. She received the Dipl.-Inform. and Dr. rer. nat. degrees in Computer Science from the Technical University of Kaiserslautern (Germany), and the Dr. Eng. degree in Production Engineering from the Federal University of Santa Catarina. She is also PMP – Project Management Professional and MPS. BR Assessor and Implementor.

R. Missfeldt Filho, is an undergraduate student of the Computer Science course at the Federal University of Santa Catarina (UFSC) and a scholarship student at the initiative Computing at Schools/INCoD/INE/UFSC.

APPENDIX A. Overview on the SE competencies taught in K-12

Reference	Learning objective(s) with respect to SE After the IU the students should be able to:	SE knowledge area(s)	SE methods/ technique(s)	SE tool(s)
(Bollin and Sabitzer, 2015)	**apply** programming skills by creating apps following SE practices.	• SW Requirements • SW Construction • SW Testing • SW Maintenance • SW Engineering Management • SW Engineering Process • SW Quality	• Pair programming • Requirement specification • SW Validation	NI
(Collofello, 2002)	**understand** the software development process. **understand** the careers of a software engineer.	• SW Requirements • SW Design • SW Engineering • SW Construction • SW Testing	• Use case • Test case • Object-oriented modeling	Rational Rose
(Corbett and Nesiba, 2015)	**apply** programming competencies by following an engineering design process including SE practices.	• SW Design • SW Construction SW Testing • SW Engineering Process	• Programming design process • Iterative process • Pseudocode • Flow diagram	NI
(De Kereki and Manataki, 2016)	**understand** and apply basic SE practices of modeling, creating, debugging, reusing computer programs.	• SW Requirements • SW Design • SW Engineering Models and Methods • SW Construction • SW Testing • SW Maintenance • SW Quality	• Debugging • Reuse • Event-driven programming	NI
(Fronza *et al.*, 2017)	**apply** skills related to an agile software development process.	• SW Requirements • SW Design • SW Construction • SW Testing • SW Engineering Process • SW Engineering Models and methods	• Storyboard • Iterative process • Brainstorming • Flow diagram • Reuse • Debugging • Agile method • Feasibility table • Mental map	GIMP

(Fronza *et al.*, 2016)	**apply** SE concepts.	• SW Requirements • SW Design • SW Engineering Models and Methods • SW Construction • SW Testing • SW Engineering Process	• V Model • Pair programming • Storyboard • Paper prototype • Unit testing • Iterative process	NI
(Fronza *et al.*, 2015)	**apply** an SE process for mobile application development.	• SW Requirements • SW Design • SW Construction • SW Testing • SW Engineering Models and Methods	• Agile method • Feasibility analysis • Iterative process • Paper prototype	NI
(Hermans and Aivaloglou, 2017)	**analyze** quality-related SE techniques.	• SW Maintenance • SW Quality	• Code smell • Debugging • Duplication • Refactoring • Naming	NI
(Köhler *et al.*, 2012)	**apply** sw-development process competencies using the waterfall model.	• SW Requirements • SW Design • SW Engineering Models and Methods • SW Construction • SW Testing • SW Engineering Process	• Waterfall model • Requirements refinement based on wireframes • Usability testing • Paper prototype • State machine diagram	NI
(Missiroli *et al.*, 2017)	**apply SE** skills using SCRUM or the Waterfall model.	• SW Requirements • SW Construction • SW Testing • SW Engineering Process • SW Engineering Models and Methods	• Scrum • Waterfall model • User story • Use case diagram	NI
(Missiroli *et al.*, 2016)	**apply** the agile methodology Extreme programming.	• SW Requirements • SW Construction • Software Testing • SW Engineering Models and Methods	• Agile method • Extreme programming • -Time boxing • User story • Pair programming • Test Driven Development	NetBeans, JUnit
(Rusu *et al.*, 2011)	**understand** the 4 types of software maintenance: adaptative, corrective, perfective and preventive.	• SW Maintenance	• Reuse (adaptive) • Corrective programming (corrective) • Exception detection (perfective) • Planning (preventive)	Educational game

(Rusu *et al.*, 2010)	**understand** the phases of the software life cycle. **understand** the roles/careers of IT professionals.	• SW Requirements • SW Design • SW Engineering Models and Methods • SW Construction • SW Testing • SW Maintenance • SW Engineering Process	• Waterfall model • Brainstorming • Control structure • Fault-based technique (test)	Educational game
(Sarkar and Bell, 2013)	**apply** acceptance tests.	• SW Testing	• Black-box test • -Acceptance test	Test case validation tool (testBedv9.html)
(Serrano and Serrano, 2013)	**apply** an SE process including requirements elicitation, requirements modeling and software validation.	• SW Requirements • SW Design • SW Testing	• Goal orientation • Scenario • Prototype • Requirements modeling	NI
(Starrett, 2007)	**apply** software modeling skills using UML.	• SW Design • SW Engineering Models and Methods	• Class diagram • State machine diagram • UML • Model Driven Development • -Model Driven Architecture	NI
(Verhoeff, 2006)	**apply** SE skills in the development of a pre-defined software project.	• SW Requirements • SW Design • SW Engineering Models and Methods • SW Construction • SW Testing • SW Maintenance • SW Configuration and Management • SW Engineering Management • SW Engineering Process • SW Quality	• Unit testing • Project review • State machine diagram • Iterative process	NI

APPENDIX B. Overview on the Characteristics of the IUs

Reference	General learning objective of the IU	General description	Education mode	Programming environment(s)	Instructional method	Instructional material	Assessment method(s)/ instrument(s)	Language(s)	License
(Bollin and Sabitzer, 2015)	Create apps to learn and review topics from other disciplines.	Students learn the basics of app programming, including database, image processing, creating apps.	In-class	App inventor	Lecture, exercise, sw project	Exercise sheets, Exemplary apps (software artifact)	NI	NI	NI
(Collofello, 2002)	Understand main SE concepts.	Students develop a sw in a simulated environment.	In-class	NI	Lecture, Simulation	Online material	NI	English	NI
(Corbett and Nesiba, 2015)	Introduce an iterative software development process to provide students with a structured procedure for addressing and solving programming problems.	Students program a Parallax Boe-Bot robot following a programming design process as part of the curriculum.	In-class	PBASIC	Lecture, sw project	NI	NI	English	NI
(De Kereki and Manataki, 2016)	Understand and apply programming fundamentals, computational thinking and the use of SE best practices.	MOOC presenting a series of challenges: how to create and remix games and animations.	Online	Scratch	Independent instruction (online course) with exercises, challenges	Videos, questionnaires, forums, research materials, sample codes (software artifact)	Quiz and peer reviews	English and Spanish	Creative Commons CC-BY-NC-SA 4.0 International License
(Fronza *et al.*, 2017)	Improve students' ability to conceptualize, understand, and utilize computational technologies. Change the perception about the career of a professional in computing.	The IU focuses on the development of animations in Scratch. Development occurs in iterations in which students are expected to plan, estimate, execute, test, and redo, if necessary.	In-class	Scratch	Lecture, exercise, sw project	Exercise sheets, sample solutions (software artifact)	Performance-based assessment and artifact-based interviews	Italian	NI

(Fronza *et al.*, 2016)	Apply current software development topics through mobile devices.	Students create a mobile application in a multidisciplinary way.	In-class	App inventor, R	Homework, exercises, problem solving, study group	Tutorial, problem overview (software artifact)	Performance-based assessment and artifact-based interviews	NI	NI
(Fronza *et al.*, 2015)	Understand computational thinking in mobile application programming	Summer school on mobile development, as a one-week experience providing lessons and hands-on experience on current mobile software development topics.	In-class	App inventor	Lecture, exercise, sw project, presentation	NI	Performance-based assessment and artifact-based interviews	NI	NI
(Hermans and Aivaloglou, 2017)	Develop the main SE programming and practices.	MOOC that teaches the creation of games.	In-class	Scratch	Independent instruction (online course)	Videos, exercises, quizzes, forum, test	Quiz e exam		Creative Commons CC-BY-NC-SA 4.0 International License
(Köhler *et al.*, 2012)	Run a software life cycle and create interest in the SE area.	Students execute all stages of an SE lifecycle by developing a quiz app for iOS devices.	In-class	NI	Goal-based scenario based learning with instructors and tutors	- Wireframes indicating important requirements for each screen (software artifact) -Interactive tutorial for programming - Tip sheet with the most important instructions for the programming language - System object model (source code) (software artifact)	NI	NI	NI
(Missiroli *et al.*, 2017)	Apply software development competences using the waterfall or agile methods (Scrum).	This IU aims to teach the waterfall model or Scrum to compare the learning results of the two approaches.	In-class	Java	Lecture, sw project	Project overview, sw requirements specification, user stories (software artifact), slides, video	Performance-based (challenge)	Italian	NI

Reference	General learning objective of the IU	General description	Education mode	Programming environment(s)	Instructional method	Instructional material	Assessment method(s)/ instrument(s)	Language(s)	License
(Missiroli *et al.*, 2016)	Apply software engineering competencies by addressing agile development practices.	Students develop a web application based on predefined user stories.	In-class	Java (NetBeans)	Lecture, sw project	User stories (software artifact)	Performance-based assessment	NI	NI
(Rusu *et al.*, 2011)	Understand sw maintenance strategies.	Students must successfully win a defense game. The goal is to prevent ants (a metaphor for insects) from crossing a map. For this, one must construct towers that shoot at them when they pass. The strategies used to attack ants represent forms of maintenance.	In-class	--	Game-based learning	Educational game	Game score, Quiz	English	NI
(Rusu *et al.*, 2010)	Understand sw life cycle processes following the waterfall model.	The students play a game aimed at the development of a website for a hypothetical company. The game is composed of 5 minigames to teach the characteristics of the phases of the waterfall model.	In-class	--	Game-based learning	Educational game	Exercises	English	NI
(Sarkar and Bell, 2013)	To apply acceptance test in which the user checks if the program does what it should do.	Students create a set of test cases for a software program that contains faults.	In-class	NI	Lecture and experience in practice,study group	-Interactive online book - Default sw program with errors (software artifact) - Flyer to describe test cases (software artifact)	NI	English	Creative Commons CC-BY-NC-SA 4.0 International License.*

(Serrano and Serrano, 2013)	(i) To improve logical reasoning (ii) to integrate/socialize secondary school students with university (iii) to apply SE concepts.	The IU teaches the development of games and robots.	In-class	Alice and Educational Robotics	Sw project	Work diaries to record the experiences gained	Peer reviews	Portuguese	NI
(Starrett, 2007)	To apply the abstract thinking process involved in modeling before programming.	Students model a software that is translated into ANSI C and compiled into the built-in microcontroller used by the Lego Robot.	In-class	ANSI C	Lecture, sw project	NI	Quiz	NI	NI
(Verhoeff, 2006)	Apply SE competencies to develop quality software.	Students develop an elevator system based on a predefined architecture and components. 5 groups of students developed a component of the system that was divided into 5 components.	In-class	Delphi Pascal	Study group, discussion, sw project	Partially pre-defined technical documentation	Performance-based assessment	NI	NI

* (http://www.csfieldguide.org.nz/en/index.html)

APPENDIX C. Overview on the Evalutions of the IUs

Reference	Research design	Factor(s) evaluated	Data collection method(s)	Sample size	Replicated studies	Data analysis method(s)	Findings
(Bollin and Sabitzer, 2015)	Case study	Aspects related to students (interest and motivation), importance of SE topics; learning difficulty, difficulties with App Inventor, product-related aspects (app quality)	Observation, open interview and questionnaire	33 students	NI	Qualitative analysis	- Despite students inexperience with computing, they were able to create good, useful apps - SE topics can be successfully introduced in high school - Computer education/ SE can be integrated with other projects/disciplines
(Collofello, 2002)	NI	Learning	NI	NI	NI	NI	Students developed the requirements using use cases. They documented the requirements using the Rational Rose tool. The research demonstrated that the curriculum can be implemented in secondary schools.
(Corbett and Nesiba, 2015)	Ad-hoc	Usefulness and ability of the process to help students meet programming goals.	Questionnaire	NI (small sample)	NI	NI	NI
(De Kereki and Manataki, 2016)	Case study	Completion rate (engagement); achievement of expectation (efficiency); interest in continuing programming; speed/ pace/ duration of the course; most valuable elements of the course; intention to recommend the course; overall experience	Questionnaire, log file (videos)	12.658 students	No	Quantitative descriptive analysis	1) 90% of the students who completed the course would like to continue to program in the future. 2) 66% of students intend to continue to program other sw as well as to know another programming language.
(Fronza *et al.*, 2017)	Quase-experimental	Efficiency of the IU teaching computational thinking	Questionnaire, creative artifacts, interview	42 students	NI	Qualitative and quantitative descriptive analysis	IU considered effective in teaching computational thinking. The agile methodology can be used to organize the development. Integration in a multidisciplinary way was possible. Additional tools and strategies are needed to complete knowledge regarding the traditional sw development process.

(Fronza *et al.*, 2016)	Ad-hoc	Performance	Artifacts created, observation	29 students	2	Qualitative analysis	Students implemented their solutions without the need for specific training in SE tools. The creation of apps helps in increasing the interest and curiosity of the students. Students have begun to recognize that IT professionals are not only programming, but they also apply SE competencies.
(Fronza *et al.*, 2015)	NI	NI	NI	NI	NI	NI	NI
(Hermans and Aivaloglou, 2017)	Case study	Difference of learning difficulty between programming concepts and SE concepts; differences in learning by age; prediction of successful completion of the course (engagement)	Quiz, log file (videos, forum), quests	2.220 students	NI	Quantitative descriptive analysis and statistical tests	1) There is no difference in performance in relation to SE learning and programming. 2) Students over the age of 12 perform better to learn operators and procedures. 3) Students who took the courses together with parents at home, who enrolled late in the course and who have low grades in the first week have a greater chance of not completing. Those who had a good participation (quizzes, forums and videos) in the first week have a greater chance.
(Köhler *et al.*, 2012)	Ad-hoc	Interest in SE	Observation	11 students	No	NI	- It was discovered that for students with no experience in computing it is better to start SE using waterfall model with short duration phases, since this model has a clear distinction of development phases - Students have shown interest in SE and computer science and are now thinking of studying it.
(Missiroli *et al.*, 2017)	Experimental	Performance and satisfaction of students and instructors	Artifacts created (code), questionnaire	160 students	7 classes in 2 schools	Quantitative descriptive analysis and statistical tests	Scrum produces usable software with better working features compared to the waterfall modell; it is also more appreciated bystudents. Teachers are more experienced with the waterfall model, which also offers more control on the development process.
(Missiroli *et al.*, 2016)	Case study	Performance, code quality, satisfaction, usability	Code analysis, questionnaire, interview	84 students e 110 teachers	4	Qualitative analysis	In general, teachers agreed on the good outcome of the experience with respect to the motivation and learning environment. Pair programming was considered the best technique. User stories did not create many advantages, but were considered useful. Test-Driven Development and Time boxing were not very well seen.

Reference	Research design	Factor(s) evaluated	Data collection method(s)	Sample size	Replicated studies	Data analysis method(s)	Findings
(Rusu *et al.*, 2011)	Ad-hoc	Learning, enjoyable	Questionnaire, quiz	18 (middle school) 10 (high school) 13 (undergraduates)	No	Quantitative descriptive analysis and qualitative analysis	Most students liked to play the game. Students realized that their strategy should change constantly when playing the defense game and that the same strategy applies to software maintenance.
(Rusu *et al.*, 2010)	Case study	Learning	Questionnaire (pre and post)	9 students	No	Qualitative analysis	Participants gained a better understanding of the software life cycle, but the maintenance phase seemed to be the confusing for the participants. The students who played the games could also to appreciate the profession of software engineering.
(Sarkar and Bell, 2013)	Ad-hoc	Strengths and weaknesses of the tool	NI	10 students	No	NI	There are opportunities for improvement with respect to the tool, such as including event tracking and analytics.
(Serrano and Serrano, 2013)	Ad-hoc	Results obtained (learning)	Self-assessment of students, Log file (work diary)	13 students	No	NI	Results demonstrate that IU contributes to socialization among students and teachers and ease of SE learning.
(Starrett, 2007)	Ad-hoc	Difficulties in learning	Observation, Quiz	NI	No	NI	Students are finding abstractions for elements in the application domain, rather than quickly starting the implementation. Data modeling concepts, such as generalization, associative classes, and reflexive associations, come naturally. Students have more difficulty with syntax than with abstraction.
(Verhoeff, 2006)	Ad-hoc	Experience, student satisfaction, SW quality, strengths and weaknesses	Artifacts created (software)	approximately 72 students	6 (3 different secondary schools)	NI	- The experience was quite successful. - Students liked the IU, which also helped them to form a better opinion on computing. - Quality of the created software needs to be improved by introducing a greater focus on modeling and testing.