# A Strategy for Assessing the Acquisition of Computational Thinking Competences: A Software Engineering Approach

Luis Corral
Universidad Autonoma de Queretaro
76000 Queretaro, Mexico
luisrcorral@gmail.com

Ilenia Fronza
Free University of Bozen/Bolzano
39100 Bolzano, Italy
Ilenia.Fronza@unibz.it

**Abstract**

Evaluating the acquisition of Computational Thinking skills can be a very complicated process, especially in a context in which, in addition to the development of the skills, the quality of an outcome product is of particular relevance. In this paper, we propose a strategy to assess the acquisition of Computational Thinking competences through eight metrics that measure the quality of a working software product. Our approach includes leveraging Block-Based Programming Languages to incorporate Software Engineering practices for a metric collection effort that delivers value for the evaluation of the quality of software product as an observable outcome of the acquisition of Computational Thinking competences.

## 1 Introduction

Computational Thinking (CT) has received attention both from Research on Education and Research in Computer Science to the point that the body of knowledge in this subject allows for systematic studies [GP13, KGK16, LM17]. Also, mainstream media has paid attention in CT as an enabler for constructing transversal skills that, through structured thinking and disciplined reasoning, can bring computational capabilities to the service of diverse professional areas of specialization. Structured thinking and disciplined reasoning, along with a sufficient command of software tools can help non-software professionals to develop simple development activities, such as designing a macro in Microsoft Office documents, writing Python or R scripts for statistical analysis, or writing HTML documents for web pages. These are illustrative examples of cases where the acquisition of CT skills can be brought to a productive setting and can be associated to an outcome software tool that boosts productivity or improves the quality of a professional environment.

Although considerable effort has been made to understand how to evaluate the effectiveness on the acquisition of CT skills [FIC17a], major assessment models miss quantitative indicators that can show the attainment and possible success of a CT skill accurately. Evaluation models give preference to a qualitative assessment viewpoint, leaving open the opportunity to understand the comprehension of the concepts quantitatively.

In this paper, we propose a compound assessment model in which the evaluation of the CT skill could be associated with Software Engineering (SE) characteristics that are observable in an outcome software product. To this end, we present a strategy to map a selection of CT dimensions into software quality assurance characteristics, illustrated by metrics that allow a quantitative indicator. The plan includes proposing case studies to run this mapping in independent populations, to shed light on how this CT assessment effort can be executed in different contexts.

The rest of this paper is structured as follows: Section 2 discusses strategies for CT assessment and CT dimensions; Section 3 presents an association between CT dimensions and SE quality characteristics; Sec-

In: I. Fronza, C. Pahl (eds.): Proceedings of the 2nd Systems of Assessments for Computational Thinking Learning Workshop (TACKLE 2019), co-located with 14th European Conference on Technology Enhanced Learning (EC-TEL 2019), 17-09-2019, published at http://ceur-ws.org.

tion 4 describes the experimental setting in which this strategy shall be applied, and Section 5 establishes expectations for future efforts and closes this work.

## 2 Assessment of Computational Thinking Skills

Assessing the learning of CT skills has been a continuous challenge that has not accomplished yet a steady level of agreement. It is easy to find a complete body of research that has focused its efforts on a number of research goals pertaining CT, mainly curriculum design and implementation of software tools.

The systematic study of Kalelioglu et al. [KGK16] shows that out of a universe of 125 examined papers, 43 articles focus on curriculum design, 34 on tool usage, and only 13 were flagged as focusing on *pedagogical frameworks*, including learning assessment. The mention to CT assessment in Grover and Pea [GP13] is as well minor, including three citations of articles whose focus is evaluation of CT skills. Nonetheless, this systematic study acknowledges the importance of methods of evaluation, and how *"without attention to assessment, CT can have little hope of making its way successfully into any K-12 curriculum"*. Finally, the systematic mapping of Lockwood and Mooney [LM17] does not consider assessment of CT as a separate research focus to study, although the concept is partially mentioned when discussed other research contributions.

### 2.1 Qualitative Strategies

Frameworks for evaluating CT are constructed depending on the needs of the context, segment and educational goals of the population (for instance K12 [SF13, MDD+14], Middle/High School [WDCK12, FIC17b] or Professionals of Education [RGPJ+16, YGGM17]). Research products in the matter show that authors do not concur in a common strategy, typically motivated by the fact that different CT assessment frameworks are implemented in distinct application environments. Each proposed method is very tailored for the instance, and the assessment strategy depends heavily on the context and population. Moreover, the evaluation models are commonly qualitative and lack of consistent metrics. Instead, questionnaires, empirical enquiries, or observations of behavioral traits are some of the preferred methods to assess the acquisition of CT skills from a *presence/absence* or *success/failure* point of view.

### 2.2 Quantitative Strategies

In the quantitative side, the work of Werner et al. proposed a more quantitative, product-focused approach that, in addition to the traditional demographical and behavioral traits, it introduced the use of characteristics of programming artifacts to evaluate students' understanding of CT notions such as abstraction, conditional logic, algorithmic thinking, and others [WDCK12]. Moreover, the study published in [SF13] attempts to create a framework to assess CT in primary grades using a quantitative approach. The study published in [MPK+08] lays the foundation to analyze aspects of a Scratch project by assessing the presence or absence of a certain programming trait such as loops, conditions, variables, etc.

Table 1: Computational Thinking Dimensions.

| Dimension | Descriptor |
|---|---|
| **Computational concepts** | Sequences |
| | Loops |
| | Events |
| | Parallelism |
| | Conditionals |
| | Operators |
| | Data |
| **Computational practices** | Iterations |
| | Testing |
| | Reusing |
| | Modularizing |
| **Computational perspectives** | Expressing |
| | Connecting |
| | Questioning |

### 2.3 Automated Tools

Tools like Dr. Scratch [MLRRG15] and Hairball [BHL+13] have had a tremendous success on evaluating block-based programming snippets and even associating them to specific CT skills, but in a approach that is language-centric to Scratch and that leaves open the opportunity to extend the approach to other languages.

To understand better what are the items of interest that are regularly evaluated in CT assessment models, it is relevant to review the work of Brennan and Resnick [BR12], who, inspired in a certain development framework, structured and organized the CT skillset that can be extended to other development tools. They distinguish common items to watch when teaching under the CT framework, and structures these items in the so-called *Computational Thinking Dimensions* shown in Table 1.

The structure of dimensions and associated descriptors are of paramount help to understand better what is the expectation in the learning outcome of a person who is receiving training in CT. Also, the organization *dimensions-descriptors* permits to outline possible characteristics that, if an outcome software product exists, can be evaluated quantitatively to determine the level of comprehension (that is, understanding and application of the concept) that the student attained after a series of explanations and exercises.

## 3 A Quantitative Quality Assurance Model for CT Skills

The model proposed by Brennan and Resnick [BR12] stops at identifying descriptors upon which measurements can be defined, leaving open the opportunity of associating such descriptors to quality characteristics that can be in practice scrutinized in the software product using a Software Engineering approach, namely the implementation of software metrics.

Software metrics have been intensely used in educational, research, and practitioner contexts to understand, measure, and assess the software product, to eventually predict the quality of the software product. The international standard ISO/IEC 25010 [ISO10] defines two dimensions of software quality:

- ***Internal/External Quality:*** Characteristics that define a view of "quality" from the standpoint of the "developer", which considers and measures only development artifacts. As ISO 25010 sheds light on strategies to evaluate relevant quality characteristics of the software product, it opens the door for proposing concrete measuring strategies or metrics, since it does not recommend how to track quality attributes accurately. This overlaps with the presence of CT descriptors, that outline a set of characteristics to measure CT success but do not recommend a metric to measure them precisely.
- ***Quality in Use:*** The degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use. This means a perspective of quality from a standpoint is of the user in her/his role of "end-user", as this dimension of quality is observable only when the final product is used in execution conditions. This approach is the most studied and implemented in previous CT assessment research since it focuses on the success of the student implementing a CT strategy, and in the *success/failure* of the outcome product from a subjective end-user viewpoint.

With a reasonably rigorous Software Engineering approach, a precious instrument to understand how CT skills are learnt and implemented is *an outcome software product.* The opportunity of analyzing it and measuring it from a development point of view makes the Internal/External Quality Assurance perspective of high interest for CT analysis. Considering as well that CT commonly leverages the possibilities offered by Block-Based Programming Languages (BBPL) such as Scratch[1] or AppInventor[2], this enables the possibility of performing software quality analysis in products developed using those frameworks. The main advantage of this approach is to offer quantitative evidence to relate an internal characteristic of the outcome software product to the acquisition of a particular CT skill.

To implement the strategy in an ordered and sequential manner, we start by placing a high-level question:

***RQ.*** *How source artifacts can be analyzed to provide an objective notion of the acquisition of CT skills?*

This question represents the main challenge of our strategy: associating the CT descriptors to source code characteristics that be measured and described as regular software metrics [FB14], following a strategy that permits to infer software quality metrics from external quality indicators [CSS14].

To assist in the identification of the proper metrics to observe the CT skill, we suggest to follow a flow roughly inspired in the GQM approach proposed by Basili et al. [BCR94, FP18]: this implies to define a goal, decompose the goal into questions, and discretize the answer to those questions in metrics that collect the necessary information to construct a solution. With this, the analysis is structured as follows:

- **Goal** (conceptual level): The overall goal that the student aims to attain, obtaining a good command of *CT skills*, understood as a collection of *CT dimensions.*
- **Questions** (operational level): Concepts to characterize the object of measurement with respect to a selected quality characteristic, in this case the *CT descriptors* that the student aims to exercise.
- **Metrics** (quantitative level): Data associated with each question to answer it numerically. With the Internal/External quality approach, such data may come from *BBPL source code characteristics.*

To conclude the analysis, it is necessary to provide a description of each CT dimension and its associated CT descriptor, to have a better understanding of the concept and be able to propose a corresponding software metric. Descriptions are taken from [BR12].

[1]https://scratch.mit.edu/
[2]http://appinventor.mit.edu/

### 3.1 Computational Concepts

Computational Concepts are traits that are common in many programming languages and that are easily mapped to features typically found in BBPLs. Computational Concepts enabler a programmer to structure and give direction to the execution flow that the developer aims to model.

- **Sequences:** A series of individual steps or instructions that can be executed by the computer. This can be observed scrutinizing how students assembly steps, for instance, by measuring the `number of consecutive blocks` for each execution flow in a project.
- **Loops:** A mechanism for running the same sequence multiple times. This can be measured counting the `number of iterative sequences` (while, for, do-while) present in a project.
- **Events:** One thing causing another thing to happen. This can be measured counting the `number of event listeners` ("when" blocks) present in a project.
- **Parallelism:** Sequences of instructions happening at the same time. This can be measured by visually inspecting and analyzing the `number of concurrent procedures` that effectively take place at the same time.
- **Conditionals:** The ability to make decisions based on certain conditions, which supports the expression of multiple outcomes. This can be observed by counting the `number of condition blocks` present in a project.
- **Operators:** Provide support for mathematical, logical, and string expressions, enabling the developer to perform manipulations. This can be measured counting the `number of operators` (arithmetic, logic, string) present in a project.
- **Data:** Data involves storing, retrieving, and updating values. This can be observed by counting the `number of variables blocks` used in a project.

Most of the Computational Concepts can be measured by performing code analysis of the component blocks, except for parallelism that requires additional analysis to determine whether more than one kernel of blocks are executed at the same moment in time.

### 3.2 Computational Practices

Computational Practices focus on the process of thinking and learning, moving beyond what students are learning to how students are learning.

- **Iterations:** The adaptive process to plan for the design, and then implementing the design in code. Although this practice can be observed counting the `number of sprints` used by a person or team to develop the project, the collection would not be product-centric.
- **Testing:** Debugging practices, commonly developed through trial and error. Much like Iterations, this practice can be tracked by the `number of test scenarios` designed to validate the product, however the metric would be more process-based and less product-centric.
- **Reusing:** The process of baselining an already-finished project and perform the necessary changes to fork it as a new product. As the practice is completely procedural, it remains out of the scope of this metric framework.
- **Abstracting/Modularizing:** Building something large by putting together collections of smaller parts, for example by creating reusable modules. This can be observed by counting the `number of stored procedures` used in a project.

As observed, Computational Practices are typically procedural and process-focused, allowing for very little room for source code analysis and data collection.

### 3.3 Computational Perspectives

Computational Perspectives describe the shifts in perspective that are observed in students when exposed to CT practices.

- **Expressing:** The ability to see computation as more than something to consume but something that can be used for design and self-expression.
- **Connecting:** The capacity of having access to new people, projects, and perspectives via networks.
- **Questioning:** The empowerment to ask questions about and with technology.

Since these descriptors fall in the area of personal analysis, we leave them out of the scope of this strategy, that aims to be product-centric.

With this analysis, we observe that mostly Computational Concepts can be inferred by SE metrics,

Table 2: Computational Thinking / Software Engineering Metrics Set.

| Dimension | Descriptor | Associated Metric |
|---|---|---|
| **Computational concepts** | Sequences | Count of consecutive blocks |
| | Loops | Number of Iterative Sequences |
| | Events | Number of Event Listeners |
| | Parallelism | Number of Concurrent Procedures |
| | Conditionals | Number of Condition Blocks |
| | Operators | Number of Operators |
| | Data | Number of Declared Variables |
| **Computational practices** | Abstracting/modularizing | Number of Stored Procedures |

while Computational Practices fall more in the scope of Software Process Analysis. Finally, Computational Perspectives shall continue being studied and analyzed from a qualitative, behavioral point of view and remain out of the scope of our model. The collection of characteristics that can be measured directly in the source software to be associated with CT concepts and practices, constructs a family of 8 metrics for CT assessment, which is illustrated in Table 2. This metric set is an abstract parametrization of CT measures that may be implemented in many ways, included visual inspection or automatic source analysis.

Although our metric set aims to be comprehensive and cover a good range of Computational Concepts and Practices, it is important to flag its limitations. The metrics shed light to note the presence/absence of a type of block (e.g., loops, event listeners) or a coding practice (e.g., procedures) in a programming project. Then, by counting the number of appearances, it conveys the extent to which the block or a coding practice is used. Since not all kind of programming projects needs to handle all types of blocks, the numbers delivered by the metrics would require a more in-depth analysis, as it would not be realistic to assume that an individual has poor CT skills when not using a block even if it is not necessary for a project. For this reason, we suggest the introduction of a *qualifier parameter* to complement the value of each metric.

### 3.4 Qualifier Parameter

To overcome this limitation, an additional parameter can be incorporated to refine the overall assessment of the quality of the values that construct a metric.

Let $M$ be a metric of the model. A parameter $Q$ can apply a value that qualifies the accuracy in which the elements of $M$ are used. The more accurately, precisely and finely the elements of the metric $M$ are used, the higher value of $Q$ shall be set, to ensure direct proportionality. The higher the value of $Q$ is, the higher the reward to the metric $M$. To establish a range for $Q$ values, we propose the following scale:

- **Outstanding (4):** The blocks are used properly and deliver additional value to the solution (e.g., it optimizes the solution).
- **Good (3):** The blocks are used properly as per the requirement of the algorithm.
- **Fair (2):** The blocks are present with no visible contribution to the solution, but without introducing an error.
- **Poor (1):** The blocks are present, and their presence impacts negatively the solution, (e.g., it injects an error).

In this way, the calculation of a qualified metric $QM$ could be set as:

$$QualifiedMetric = Metric * Qualification \tag{1}$$

To understand it better, let us suppose a case in which the metric $M$ represents the number of iterative sequences (loops) in a project. Then, let us suppose two competing projects $A$ and $B$, with the same number of loops (for instance, 5).

In project $A$, all the loops are correctly set and contribute effectively to the solution of the problem. The qualification of these loops would be Good, that is, 3.

In project $B$, some of the loops are misplaced and contribute marginally to the solution of the problem. The qualification of these loops would be Fair, that is, 2.

Per equation (1), the qualified metric $QM$ for both projects is:

$QM_A = 5 * 3$

$QM_A = 15$

And,

$QM_B = 5 * 2$

$QM_B = 10$

Although both projects have the same number of loops and hence the same value of the illustrative metric $M$, the qualifier $Q$ allows the calculation of two very distinct values, rewarding the project in which the assessed characteristic was better used, or penalizing a practice that instead of contributing to the solution, opens an opportunity of error.

The introduction of the qualifier parameter permits a better association of the metric value with its actual contribution to the overall assessment of the CT quality. Nonetheless, it poses an additional challenge in the metric management strategy to determine who and how should execute the quality assessment to assign a rigorous and accurate qualifier value, without being subjective.

## 4 Case Studies

To have an initial approach as to how the proposed model can deliver insightful information on the way that a population is learning CT skills, we propose to set up training experiences of Block-Based Programming Languages (BBPL) addressed to two different populations of non-expert subjects. Programming environments like these are often used as the basis for CT assessment, to have a steady environment for evaluating tasks or measure products for aspects of algorithmic thinking, abstraction, and modelling.

The BBPL that will be taught is Thunkable[3], a mobile app development language that permits to design user interfaces and executable programs using blocks of code (Figure 1). To open the possibilities of the case studies, Thunkable was selected as it respects the BBPL programming paradigm, and permits multiplatform development for Android and iOS. The teaching staff will be the same for both courses. The metrics will be collected manually, and the criteria to qualify the metrics will be set by the teaching staff.

### 4.1 Group 1: A Bootcamp for Middle Schools

We will introduce our metric set to analyze the outcome products of a one-week course that delivers hands-on experience in software development for mobile devices. This bootcamp targets high school students with no experience in software development. The age segment of participants spans from 15 to 18 years old. The length of the course is 20 hours of hands-on training, taught in 5 sessions of 4 hours (Monday to Friday). The course will take place in Italy, in a state University.

The course offers an overview on problem-solving based on CT principles. Then, BBPL tools are introduced to represent and execute the algorithms. Participants are required to design and implement a software project working in teams of three students. The project is handed in by the end of the course and will serve as outcome product for code analysis and metric calculation.

### 4.2 Group 2: A Graduate Course for Fine Arts

The experience will be replicated in a postgraduate course that will be offered in a Faculty of Fine Arts in Mexico. The course is addressed to graduate students who have earned a Bachelor's Degree in Arts previously. The age segment of participants spans from 21 to 35 years old. The course will be carried out in a state University in Mexico.

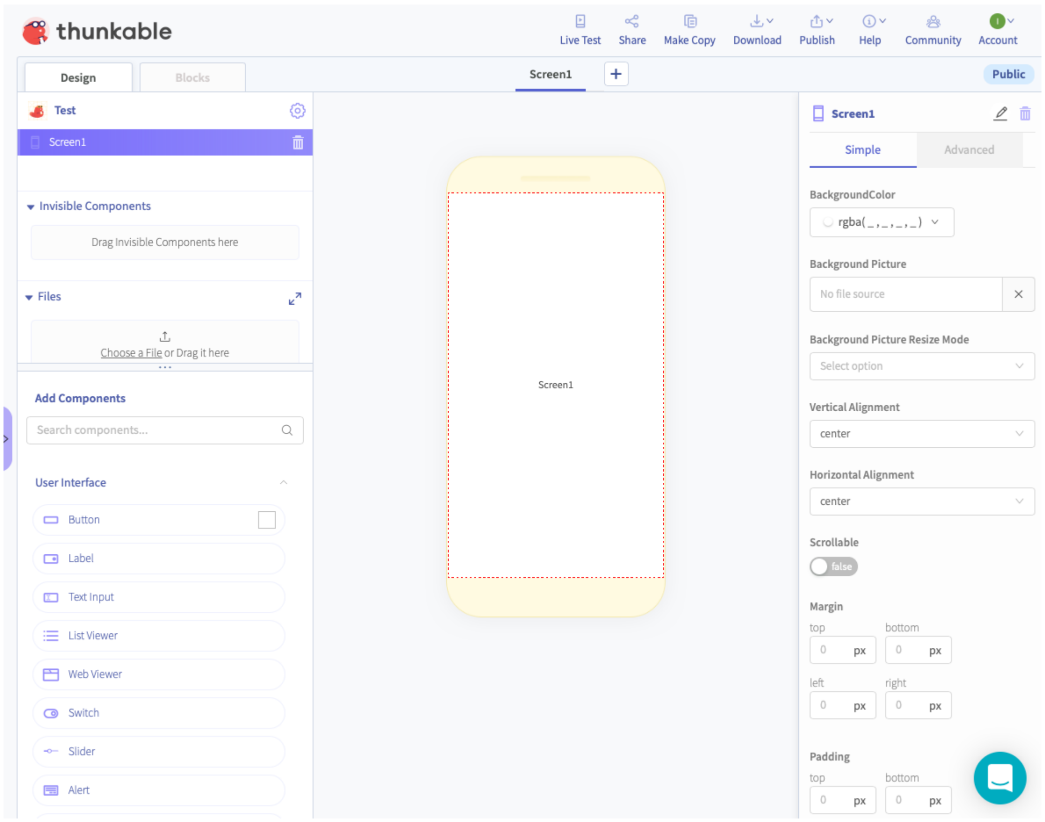
The population does not have experience in software development. The length of the course is 40 hours of theoretical and practical study, taught in sessions of 2 hours. The structure of the course will be the same of Group 1. For replication purposes, students will be as well required to design and execute a software project working in teams of three students, and the project and will be analyzed based on the proposed metric set.

## 5 Future Work and Closing Remarks

In this paper, we introduce a strategy to assess the acquisition of CT competences based on an association of CT dimensions with software code quality metrics. We envision the execution of BBPL training courses with the same content and structure, addressed to two non-expert populations, that can deliver insightful metrics collected using our model. This product-centric approach to evaluate the success of CT skills is based on eight metrics that represent CT dimensions. We expect to follow up this work by collecting a dataset that permits deepening in the analysis to strengthen the discussion to ensure a quantitative, population-agnostic CT assessment practice.

There is still much work to do to draw conclusions in the underlying ideas of this work. As future work, the case studies of Section 4 need to be implemented on top of a robust design of experiment, in which the technical coverage of the two settings is comparable so that the collection of outcome products in the two settings is similar too. Moreover, an essential next step is to determine "acceptance" ranges, thresholds and usage traits to evaluate the success of specific skills based on the value delivered by a metric. Additionally, several validity threats must be borne in mind to ensure a successful implementation of the strategy. We can identify two major ones: The manual calculation of the metrics, and the subjective assignment of the qualifier value.

[3]https://www.thunkable.com/



BackgroundColor: rgba(_,_,_,_)
Background Picture: No file source
Background Picture Resize Mode: Select option
Vertical Alignment: center
Horizontal Alignment: center
Scrollable: false
Margin: top 0 px, bottom 0 px, left 0 px, right 0 px
Padding: top 0 px, bottom 0 px

Figure 1: User interface of Thunkable.

This work is an initial step to connect Software Engineering practices for the evaluation of the quality of the software product as an observable outcome of the acquisition of Computational Thinking competences.

## References

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[BHL+13] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 215–220. ACM, 2013.

[BR12] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *In AERA 2012*, 2012.

[CSS14] Luis Corral, Alberto Sillitti, and Giancarlo Succi. Defining relevant software quality characteristics from publishing policies of mobile app stores. In *International Conference on Mobile Web and Information Systems*, pages 205–217. Springer, 2014.

[FB14] Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach*. CRC press, 2014.

[FIC17a] Ilenia Fronza, Nabil El Ioini, and Luis Corral. Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education (TOCE)*, 17(4):19, 2017.

[FIC17b] Ilenia Fronza, Nabil El Ioini, and Luis Corral. Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education (TOCE)*, 17(4):19, 2017.

[FP18] Ilenia Fronza and Claus Pahl. Envisioning a computational thinking assessment tool. In *CC-TEL/TACKLE@ EC-TEL*, 2018.

[GP13] Shuchi Grover and Roy Pea. Computational thinking in k–12: A review of the state of the field. *Educational researcher*, 42(1):38–43, 2013.

[ISO10] ISO/IEC. Iso/iec 25010 system and software quality models. Technical report, 2010.

[KGK16] Filiz Kalelioglu, Yasemin Gülbahar, and Volkan Kukul. A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, 4(3):583, 2016.

[LM17] James Lockwood and Aidan Mooney. Computational thinking in education: Where does it fit? a systematic literary review. *arXiv preprint arXiv:1703.07659*, 2017.

[MDD$^+$14] Linda Mannila, Valentina Dagiene, Barbara Demo, Natasa Grgurina, Claudio Mirolo, Lennart Rolandsson, and Amber Settle. Computational thinking in k-9 education. In *Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference*, pages 1–29. ACM, 2014.

[MLRRG15] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46):1–23, 2015.

[MPK$^+$08] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. Programming by choice: Urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 367–371, New York, NY, USA, 2008. ACM.

[RGPJ$^+$16] A Rees, Francisco J García-Peñalvo, I Jormanainen, M Tuul, and D Reimann. An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers. 2016.

[SF13] Linda Seiter and Brendan Foreman. Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 59–66. ACM, 2013.

[WDCK12] Linda Werner, Jill Denner, Shannon Campe, and Damon Chizuru Kawamoto. The fairy performance assessment: measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 215–220. ACM, 2012.

[YGGM17] Aman Yadav, Sarah Gretter, Jon Good, and Tamika McLean. Computational thinking in teacher education. In *Emerging research, practice, and policy on computational thinking*, pages 205–220. Springer, 2017.