JOURNAL OF LEARNING ANALYTICS
SOLAR SOCIETY for LEARNING ANALYTICS RESEARCH
Volume 11(2), 157–173. https://doi.org/10.18608/jla.2024.8291

# A Method for Developing Process-Based Assessments for Computational Thinking Tasks

Sohum Bhatt[1], Katrien Verbert[2] and Wim Van Den Noortgate[3]

**Abstract**
Computational thinking (CT) is a concept of growing importance to pre-university education. Yet, CT is often assessed through results, rather than by looking at the CT process itself. Process-based assessments, or assessments that model how a student completed a task, could instead investigate the process of CT as a formative assessment. In this work, we proposed an approach for developing process-based assessments using constructionist tasks specifically for CT assessment in K–12 contexts, with a focus on directly connecting programming artifacts to aspects of CT. We then illustrated such an assessment with 29 students who ranged in CT and programming experience. These students completed both a constructionist task and a traditional CT assessment. Data from the constructionist task was used to build a process-based assessment and results were compared between the two assessment methods. The process-based assessment produced groups of students who differed in their approach to the task with varying levels of success. However, there was no difference between groups of students in the scores on the traditional CT assessment. Process-based assessment from our approach may be useful as formative assessment to give process feedback, localized to the task given to students.

**Notes for Practice**

- A method of developing process-based assessments for constructionist tasks in computational thinking is proposed
- Process-based assessments can give insights into different methods of computational thinking use
- Feedback from process-based assessments may be task-specific
- The approach needs to be validated further on a bigger sample

**Keywords:** Process-based assessments, learning analytics, computational thinking, formative assessment

**Submitted:** 07/11/2023 — **Accepted:** 12/06/2024 — **Published:** 25/07/2024

*Corresponding author [1]Email: sohummandar.bhatt@kuleuven.be Address: Faculty of Psychology and Educational Sciences, KU Leuven and IMEC Research Group ITEC KU Leuven Campus Kulak Kortrijk, E. Sabbelaan 51, 8500 Kortrijk, Belgium. ORCID iD: https://orcid.org/0000-0002-8917-292X*
*[2]Email: katrien.verbert@kuleuven.be Address: Human and Computer Interaction Group, Department of Computer Science, KU Leuven, Celestijnenlaan 200A, Heverlee B 3001, Belgium. ORCID iD: https://orcid.org/0000-0001-6699-7710*
*[3]Email: wim.vandennoortgate@kuleuven.be Address: Faculty of Psychology and Educational Sciences, KU Leuven and IMEC Research Group ITEC KU Leuven Campus Kulak Kortrijk, E. Sabbelaan 51, 8500 Kortrijk, Belgium. ORCID iD: https://orcid.org/0000-0003-4011-219X*

## 1. Introduction

With the rise in personal computing, the growth of the internet, and the integration of computer technologies in most jobs, the skills to effectively utilize technology become increasingly important. Some of these skills are conceptualized as computational thinking (CT). The definition of CT is an often-debated topic, but generally, CT can be seen as the ability to solve problems using concepts from computer science (Wing, 2006).

Education in CT has been growing, fuelled by calls as to its importance in future society (Grover & Pea, 2013). Researchers in particular advocate for the use of constructionist education as a means of teaching CT (Lodi & Martini, 2021; Lye & Koh, 2014). Traditional education uses lecture-based teaching, with assignments that have a correct or incorrect answer that tests learning concepts (Lye & Koh, 2014). Constructionism is a form of education that stresses active learning through the construction of personal, hands-on projects, termed constructionist tasks in this paper (Lodi & Martini, 2021). However, with the recent focus on CT and the push for constructionist education, there is an upcoming debate on how to best assess CT. Traditionally, teachers use CT assessments comprised of multiple-choice questions or open-ended questions graded for correctness of content for both summative (i.e., aimed at measuring what has been learned) and formative (i.e.,

aimed at collecting information to improve student learning) assessments (Tang et al., 2020). These are termed traditional assessments. Portfolio assessments are also used regularly, but researchers have noted that they largely ignore the learning process and processes of use that are important for CT (Fields et al., 2019). For CT taught with constructionist tasks, assessment is slow to administer and grade. To account for the difficulties in CT assessment, we propose using process-based assessments, a growing trend in computer science research. Process-based assessments use the artifacts generated during a project to model how students complete a task. With this model, researchers and teachers can assess how well a student uses a particular skill. However, prior approaches in process-based assessments have not yet been fully evaluated in K–12. Some previous research has used K–12 populations but without connecting their findings to CT and used either an online sample or a sample of multiple K–12 classes in one study (Berland et al., 2013; Jiang et al., 2022).

This study aims to present an approach to developing process-based assessments for individual constructionist tasks in CT education. This study also aims to illustrate the possibilities of such an assessment in a small sample of K–12 students. In this work, we conceptualize constructionist tasks in CT as projects that can be completed with diverse methodologies but create artifacts with similar functions. In addition, we answer two secondary questions:

1. What CT processes can we observe during a project task?
2. How do the results of a process-based assessment compare to a traditional assessment? Studying the association will shed light on the criterion validity of the new approach.

This study contributes a methodology to produce process-based assessments for constructionist tasks in CT that can be applied to programming projects or other integrated projects. It begins to analyze CT by assessing the process, as recommended by Fields et al. (2019). This study will also contribute to the literature by exploring the use of process-based assessments in K–12 educational settings, rather than in online or university settings, which have been studied before (Blikstein et al., 2014; Jiang et al., 2022).

Our assessment approach follows previous work on process-based assessments. We count CT usage and line numbers to compare code instances. Similar counts define more similar codes. This measure is used for hidden Markov models to define different states among students. From this hidden Markov model, paths through hidden states are reconstructed for each student. These paths are compared with dynamic time warping and clustered to produce groups of paths. The groups of paths may indicate a common methodology for the use of CT, although this requires more research.

We demonstrate our assessment approach with a sample of K–12 students who first completed a traditional CT assessment. For the proposed assessment approach, students then worked to complete a coding project in Python to make a two-player game of tic-tac-toe. Following the approach, we analyzed if there was a significant difference between student path groups in CT ability as measured by the traditional assessment as a means to explore criterion validity.

In summary, our process-based assessment approach emphasizes anchoring similarity measures in explicit uses of CT found in programming tasks, such as the use of “for” and “while” loops as active uses of a student’s algorithmic thinking ability. The student paths and clusters observed in this example are directly connected to CT constructs, explore how students use CT in a new challenge, and are in line with prior research in process-based assessments. With further research, our approach to process-based assessment for CT may be useful as a formative assessment to provide strong process feedback to students as recommended by constructionist theory. For researchers, our approach gives a more direct look at CT use in action as opposed to portfolio or other assessments. However, in our sample, we found that the assessment produced by our approach does not agree with the CT ability as measured in traditional assessments.

## 2. Related Work

We now discuss the constructs and prior research that guide our work.

### 2.1. Computational Thinking

CT as a concept is relatively ill-defined. The earliest definition of CT is the linking of programming and cognitive skills (Papert, 1980). In this form, CT is seen as a means of using programming to construct abstract mental models (Lodi & Martini, 2021). For example, students who learn geometry can learn the areas of a square by experimenting on filling differently sized squares using programming. This allows students to develop their own intuitions before learning the underlying equations. A more recent definition describes CT as the way to define and solve problems using concepts originating from computer science (Wing, 2006). As such, CT is not programming, but rather the broader set of cognitive strategies that utilize the strength of a computer. Some skills within CT include the use of loops, parallelism, conditionals, iterations, and abstraction to solve problems, and decomposition by questioning and connecting information to break down a problem (Brennan & Resnick, 2012). However, there is a variety of conceptualizations of CT designed by researchers, each focusing on its own set of subskills (Dong et al., 2019). In this work, we consider CT as the application of computer science skills to solve problems, while decomposing CT into pattern recognition, abstraction, decomposition, and algorithms using the PRADA framework (Dong et al., 2019).

Despite the variety of conceptualizations and definitions, governments and non-profit bodies have increasingly advocated for CT as a subject for education, such as being included in multiple curricula reforms (National Research Council, 2013; Weintrop et al., 2021; Bocconi et al., 2022). This rise of educational interest in CT has led to questions on how to best assess it since a variety of methodologies and concerns exist (Weintrop et al., 2021). Traditional assessments in CT are multiple-choice or open-ended questions graded for correctness of content. However, for CT, traditional assessment does not reflect the learning process or the processes of how a student uses CT (termed use processes) needed for effective individual growth in CT (Fields et al., 2019).

Beyond these traditional assessments, a variety of assessments for CT is used in research, with options such as interviews, surveys, and portfolio-based assessments (Tang et al., 2020). From these options, only portfolio-based assessments are likely to be used in the classroom for pure assessment by teachers. One example of a portfolio measure in CT literature is the project portfolio analysis used by Scratch team, which scrapes a user's project portfolio to produce a heatmap of the blocks used in every project (Brennan & Resnick, 2012). For other portfolio assessments, students complete a project and are graded on a rubric that assesses different measures of CT use (Tang et al., 2020). However, portfolio measures also do not directly account for the learning process that may be important to CT. In addition, portfolio measures often fail to investigate the use processes of CT in a task, which may be of interest to researchers and teachers aiming to provide more direct feedback.

Most of these assessments have been used for summative assessment, although formative assessments are growing (Bennett, 2011). Formative assessments are important for the delivery of CT education, providing feedback for both the student and teacher with which to provide actionable advice (Grover, 2021). According to Grover's framework for formative assessment in K–12 computer science education (including CT), formative assessments share learning goals with students, demonstrate how to achieve learning goals, specify or investigate common errors, help control frustration, and provide advice to students for better understanding of materials. Few formative assessments have been developed for CT, and most use traditional modalities, meaning graded for correctness (Bonner et al., 2021; Grover, 2021). But, in addition to previously mentioned issues, formative traditional assessments do not demonstrate the intermediate steps necessary to achieve learning goals. Teachers can help students achieve learning goals by providing feedback from the results of formative assessments, but actionable feedback from traditional formative assessments may not explain the processes used to achieve learning goals. A solution to various issues with assessment in CT is the use of process-based assessments (Berland et al., 2013; Blikstein et al., 2014).

## 2.2. Process-Based Assessments

Process-based assessments model the approach a student takes to complete the assessment using the artifacts of student progress throughout a task. Process-based assessments can be automatic, can use the student's work, and can model the CT use of an individual student or group. In contrast to traditional assessments, process-based assessments may also provide useful process feedback, i.e., feedback on how to improve rather than simply on the correctness of the solution (Piech et al., 2015; Sedrakyan et al., 2016). Most work in process-based assessments has been focused on beginner computer science education in university, specifically programming courses (using the codes produced by the students as the input artifacts); however, a similar approach might be used in CT projects, particularly those that use some form of coding.

In computer science, process-based assessments are in their nascency. Despite this, there are common methodologies to create a process-based assessment. Process-based assessments in computer science follow three steps. First, the similarity between programs must be decided. This task is like establishing word embeddings in natural language processing (Almeida & Xexéo, 2019). Next, paths must be found through the changes in programs. This is usually done on a student level, but group-level paths through an assignment have provided information that aids in the interpretation of student paths (Blikstein et al., 2014; Jiang et al., 2022). Finally, the similarity between different paths of an assignment must be defined. We will now discuss the methodologies and results of previous process-based assessments.

One of the first process studies found differing patterns of coding in an introductory university course using NetLogo (Blikstein, 2011). These patterns were dependent on the prior experience of students (Blikstein, 2011). Blikstein and colleagues expanded this work in 2014. In this study, student coding was grouped by similarity using a combination of abstract syntax tree (AST) similarity and program call similarity. ASTs are the underlying representations of the syntax used to compile and run a program, and therefore can be used to compare programs (Neamtiu et al., 2005). From these code similarities, hidden Markov models were created for the entire group (a single university class) and all students individually to discover latent events and paths through events. The similarity between these student paths was defined as the probability that one student's hidden Markov model could generate another student's hidden Markov model, and this similarity was used to cluster student paths. The group paths through the assignment indicated that there were difficult states that some students remained in for many updates; Blikstein and colleagues interpreted these students as less efficient than others. The student path groups represented different approaches taken on the same assignment, and these groups correlated with grades on a traditional midterm assessment (Blikstein et al., 2014). However, this work was primarily focused on programming ability, not on CT. In contrast, little work has been performed on CT process assessment. The earliest work attempting to

connect programming patterns to CT was Berland et al.'s (2013) work on modelling high school students' programming work from a STEM-focused summer camp using a block-based programming environment, meaning premade code was visualized as blocks and could be connected, copied, or moved around. Here, code similarity was based on simple primitives found in the block-based programming language. These programs were then clustered into categories, then these clusters were used to describe the student path, charting three distinct phases: 1) exploring the language available to them, 2) tinkering with possible solutions, and 3) refining solutions to the game. This path was termed the EXTIRE pathway. Importantly, the EXTIRE pathway described all student paths through the assignment, despite the complete freedom of students to create their own solutions. However, this study did not investigate subgroups or student path groups within the EXTIRE pathway. Despite focusing on the thought processes behind programming, this study did not connect ideas of tinkering to CT concepts. A similar work by Jiang et al. (2022) studied programming paths for block-based programming languages in a large sample taken from the Hour of Code program. Block-based programming languages are commonly used to teach CT; however, this study focused on the pure assessment of programming skills in the block-based language, without any connections to CT concepts or subskills (Jiang et al., 2022; Lye & Koh, 2014). The paths found in this work were different than those found in previous works, focusing on the effort exerted by a student, rather than differing approaches or experimentation. Regardless, this study showed that, again, students take similar paths to find the solution, and that despite failures in a task, students learned and improved on the next task.

Beyond purely process-based assessments, there are other learning analytic approaches to understanding and modelling processes. One recent approach investigates a specific process of programming: debugging. In a recent study, inefficient and efficient episodes of debugging were compared to the differences in debugging strategies. Inefficient episodes of debugging were times when students restarted the debugging process by going back to their original code, whereas in efficient episodes, students made gradual progress until fixing the bug. These debugging strategies and associated indicators were found from previous literature (Liu & Paquette, 2023). Using logistic regression, researchers found that when students used certain debugging strategies, they were more likely to have an efficient debugging session. Another broader approach is modelling the problem-solving processes of students during productive failure (Hartmann et al., 2022). In this study, think-aloud procedures and screen recordings were coded to discretize problem-solving activities. These activities came from previous research and the researchers' knowledge of the problems that students needed to solve. With these activities, a model of the process used during productive failure was created. For deeper analyses, high and low performing groups within the sample were split and models that differed on the strengths and presence of different activities were found.

However, despite similarities in methodology for all investigations of process, the foundational connections of the analytics to the instruments in question are unclear. Few studies in process-based assessments directly connect their analyses and findings to concepts like algorithmic thinking. This may be because the relationship between CT and programming is not clear and is often debated (Tikva & Tambouris, 2021). Even so, some attempts have been made to map the connections between CT concepts and programming concepts, but these connections are used to define CT in terms of programming, rather than view CT as a broader skill used in programming (Brennan & Resnick, 2012; Shute et al., 2017). We believe that this limits and constrains CT to being just a method of teaching programming. In addition, the approaches used in connecting programming measures to CT may also cause difficulty for K–12 teachers to use the process-based assessments previously researched. To use previous process-based assessments, teachers need tremendous domain knowledge of both programming and CT. However, given the novelty of CT as a subject in curricula around the world, teachers are not likely to be technical experts in CT or in programming, thus requiring clear explanations and connections to curricula that may be difficult to achieve with previous process-based assessments. With the novelty of CT in curricula, interpretability is important for teachers to be able to use process-based assessments in their lessons (Rosé et al., 2019). Another difficulty with the mapping of actions to CT is that there are indications that applying CT is task-dependent (Boom et al., 2022). One study found that even though higher CT skills led to higher programming abilities, indicators of CT ability depended on the task given and the programming language used (Boom et al., 2022). This means that there are no general indications of CT ability when broadly using programming; rather high CT skills can be seen by the ability of the student to adapt and utilize programming concepts within a particular task. Therefore, it is possible that all process-based assessments for CT may only assess skills within the task itself. In addition, CT is increasingly integrated into curricula beyond simple programming courses (Dong et al., 2019; Kite & Park, 2023). In these expanded science or math contexts outside of curricula, there are further indications of the task-specific nature of CT application, with differing patterns of CT use in STEM professionals depending on how theoretical or experimental their work is (Beheshti, 2017). Previous process-based assessments also do not account for these expanded contexts, and so their implementation in K–12 is likely to be limited.

Another important consideration is that it is unknown how much information can be learned from process-based assessments in K–12 contexts, with smaller class sizes than previously studied. Previous studies had sample sizes ranging from 53 students to 263,569 students (Berland et al., 2013; Jiang et al., 2022). This is larger than K–12 contexts, where CT is taught in small classes (i.e., an average of 26 students) led by a teacher (OECD, 2020; Sands et al., 2018). Therefore, it is unknown whether a process-based assessment based on individual classes in K–12 education (i.e., a small sample size) would produce meaningful interpretations.

### 2.3. Constructionism

CT and constructionism are often connected, as both concepts originate from Seymour Papert (Lodi & Martini, 2021; Papert, 1980). CT and constructionism both use computers and programming to teach basic skills related to computer science (Lodi & Martini, 2021). CT and constructionism also suppose that these basic skills in computer science transfer and apply to contexts broader than just computer science (Lodi & Martini, 2021). However, CT has been conceptualized as a new subject or topic in education. As such, CT education uses diverse educational approaches, such as projects, but is commonly taught with traditional group-based instruction. In contrast, constructionism is conceptualized as a different form of education. Constructionism espouses the belief that the act of construction must be personal to the student for learning to take place (Lodi & Martini, 2021; Papavlasopoulou et al., 2019). This takes the form of more project-based learning, wherein students are given a task that can be completed (with skills that they must learn) based on their interests. Importantly, these tasks are often open-ended. Many approaches to teaching CT have their basis in constructionist theory, with many researchers and stakeholders advocating for more constructionist tasks and educational approaches to teaching and learning with CT (Lodi & Martini, 2021; Lye & Koh, 2014).

Yet, assessment is a well-identified problem with constructionist tasks. Initially, assessment for constructionist tasks was based on critique of student work by peers and instructors, as is done in art or writing classes (Berland et al., 2014). Another approach to assessment in constructionist environments is to describe and reflect on what a student has learned through the development of a project (e.g., Oliver et al., 2021; Roque & Tamashiro, 2022). The description of a learning process is often regarded as valuable information for student reflection and feedback in constructionist theory (Berland et al., 2014). In essence, many assessments in constructionist theory are formative, rather than summative, aimed at providing feedback from which a student can learn. This is because constructionist theory resists the categorization of students into high or low achievers (Berland et al., 2014). However, especially as formative assessment, critiques and the many descriptive techniques for constructionist assignments are slow, and teachers are often required to assess repeatedly throughout a course (Berland et al., 2014). Here, learning analytics have been suggested to add fine-grained detail for the description of student learning processes, with a focus on understanding the strategies used to construct, problem solve, and explore in constructionist tasks (Berland et al., 2014). Process-based assessments have been suggested for their ability to understand student processes and provide valuable process feedback, though again, the focus is formative. But, even so, student direction and personalization are paramount, and the solution space of constructionist tasks is extremely large, so process-based assessments are likely to be most valuable in describing how core skills are used throughout the project.

In previous works in process-based assessments, the solutions for tasks given to students were more constrained than the purely open-ended problems envisioned by constructionist proponents. This may be due to the focus on programming in previous research. However, despite this, the approaches for solutions are generally unlimited, which is closer to the problems envisioned by constructionist theory. Due to the difficulty in connecting programming to CT constructs, we design our approach for CT assignments using constructionist methodologies (i.e., that can be solved with multiple approaches), but with solutions similar in function to connect programming more readily to CT. We assume that as more tasks are made with our approach, students can eventually choose tasks that interest them, thus fulfilling the push for more personal tasks in line with constructionist theory.

## 3. Methods

We now discuss our approach to develop process-based assessments for CT in detail. Afterward, we discuss the study protocol used to demonstrate the process-based assessment produced by our procedure and compare it to a traditional assessment.

### 3.1. Approach for CT Process-Based Assessment

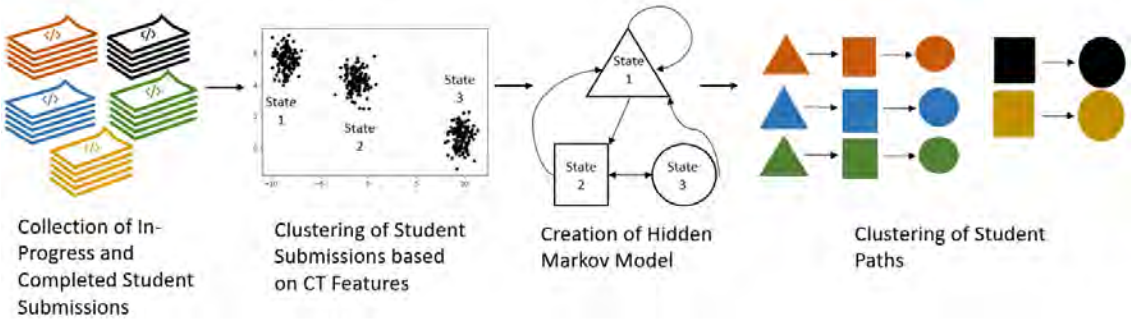


**Figure 1.** Visual representation of process-based assessment approach.
Colours represent individual students and shapes represent hidden stages.

Our approach for developing process-based assessments in CT follows constructionist methodologies. The intention is to create a simple process that can be followed by teachers and researchers to develop process-based assessments for tasks to be completed by groups of students. In this way, teachers and researchers can develop process-based assessments for more personal, meaningful projects. As such, an important caveat for these process-based assessments is that each is bound to the task given to students and assesses how well a student or group uses CT skills within that task.

Here is a summary of our process: In the first, most crucial step, teachers or assessment developers decide how CT is used in a task, and with that definition, define measurable or observable indicators of CT use. These indicators can be specific functions used, for example from programming, or the act of reviewing previous iterations for debugging as in Fields et al. (2019). It is important to note that the focus in this step is understanding and discretizing ways CT can be used, rather than marking the steps needed to complete a project. Comparing the number of times and ways that CT is used in a project is a measure of similarity between the work of students. The next step in our approach is the generation of a hidden Markov model for all students. A hidden Markov model is a sequential probabilistic model that aims to chart the progress of hidden states using only the observable states to surmise the hidden states (Rabiner, 1989). This model is created by enumerating the number of states, calculating the probability that an observation can be found from a given state (emission probability), and calculating the probability that a state transitions to a new state (transmission probability). Next, this hidden Markov model can be used to generate the sequence of states a student used to complete an assignment. With all sequences of states, clusters of student paths can be created for comparison. A visual overview of this approach can be seen in Figure 1. We now discuss each of these steps in detail using our task to illustrate the approach.

To define indicators of CT use in our task, we first solved the task as simply as possible to understand where CT is used. We then reviewed for which problems CT was used and which code fragments were used to solve them. We finally connected how and where code was used to CT subskills. Theoretically, this step can be performed without solving the task, with experts and teachers relying on their knowledge and training to connect subskills to code. In this work, we rely on the PRADA framework to define CT subskills in K–12 schools (Dong et al., 2019). The PRADA framework defines the subskills of CT as Pattern Recognition, Abstraction, Decomposition, and Algorithms. This framework is designed to allow teachers to delineate the subskills of CT for each assignment and has been used to operationalize CT in their specific domains (Dong et al., 2019). The PRADA framework was chosen for its deliberate rejection of coding components in operationalizing CT, along with its focus on clearly separating components of CT (Dong et al., 2019). It was also chosen because its definition of CT relates closely to governmental understanding of CT in K–12 (e.g., Bocconi et al., 2022). In comparison to the definition of CT used by the Scratch team from MIT, the PRADA framework focuses on computational concepts, rather than computational practices found in coding or computational perspectives developed because of learning CT (Brennan & Resnick, 2012). Our assignment for this work is a game of tic-tac-toe within a list of lists, discussed further below. Tic-tac-toe is a common method for teaching CT in K–12 scenarios (Lee et al., 2014; Ma et al., 2022; Silapachote & Srisuphab, 2016). We now describe each subskill of CT within the PRADA framework and how these subskills are marked within our task.

Pattern recognition is defined as the ability to find patterns in data. In the PRADA framework, pattern recognition is specifically focused on describing patterns within a context (Dong et al., 2019). As such, in tic-tac-toe, how a player wins can be described as a pattern. To win tic-tac-toe, a player must have a line of their game pieces horizontally, vertically, or diagonally. In this task, we can use the number of times a student codes the program to check the board as an indicator that a student recognizes the patterns involved. These are recognized aspects of pattern recognition in students' learning CT through tic-tac-toe (Lee et al., 2014). In our learning environment with Python, to check the board, a student must call the name of the board and the index of the position to be checked. This board check provides markers that can be counted. A more advanced student might manipulate the board or store game pieces in a temporary location for comparison. To capture the actions of these more advanced students, we count the number of times a student uses list functions as a marker for pattern recognition as well.

Abstraction is the subskill that focuses on using the principles underlying patterns to solve problems, and as such, abstraction and pattern recognition are intertwined in this assignment. List and board checks mark how a student uses code to account for patterns, and, as such, these indicators serve to capture abstraction, as without abstraction, a student would be unable to account for win conditions. Similarly, counting the number of uses of Boolean logic can serve as another indicator for abstraction. Boolean logic can test for equality, and using tests for equality demonstrates that a student understands how and when to compare for win conditions, i.e., uses abstraction to solve this challenge. The count of Boolean logic is only regarded as an indicator for abstraction as Boolean logic in this assignment does not directly relate to the pattern of win conditions; rather, Boolean logic is used as an abstraction to check for win conditions in code. These aspects have been found in other investigations of the use of tic-tac-toe to teach CT (Lee et al., 2014; Ma et al., 2022).

Decomposition is poorly defined, but naïvely, it is the ability to break a problem down into its components. To operationalize decomposition, we use the framework established by Rich et al. (2019), which states that functions used in programming are instruments of decomposition: "A program is often broken down into a series of objects related by the functions that they perform, and how that relates those objects to other objects in the system" (Rich et al., 2019, p. 419). This is termed functional decomposition (Rich et al., 2019). In this framework, functions are defined by the objects they

interact with and the actions performed on the objects. Thus, when students use functions in their code, they are using functional decomposition to compose their program. Therefore, we count the uses of student-defined functions and basic functions (e.g., return or input, termed actions), as indicators of decomposition. As such, while using functions can also be considered as an example of abstraction, student defined functions and uses of actions are considered functional decomposition in practice, such as printing when someone has won, or creating a function to simplify one player's actions. This is in line with other investigations of decomposition for tic-tac-toe. In these other investigations, students break the game down into components, such as placing a marker or checking for victory, which we expect to be completed with functions (Lee et al., 2014; Ma et al., 2022). In contrast, we and other investigations of CT and tic-tac-toe view abstraction as the ability to use smaller pieces to correctly capture a pattern in question (Ma et al., 2022).

Algorithms, or the use of a set of instructions to solve a problem, is easily found in how students use control flow statements in their programs, such as if, else, for, or while. Control flow has been well studied as a part of algorithmic thinking, providing a simple method to mark algorithm usage (Angeli, 2022; Wong & Jiang, 2018). The use of control flow in tic-tac-toe is the main way to program the rules of the game, a common aspect of algorithmic thinking in other works in CT and tic-tac-toe (Lee et al., 2014).

Beyond the PRADA CT skills, we count the number of lines used in each program as part of our similarity measure. The number of lines comes from Berland et al. (2013), who initially counted the number of codes used. This measure is translated to lines for Python. They use lines/codes as a naïve measure of effort. The features used are summarized below in Table 1. Again, as a caveat, these measures are based on the task itself, and do not generalize to other tasks. It is not the intention of this work to use the produced process-based assessment as summative, but rather to demonstrate the abilities of process-based assessments while providing a simple method of development for non-experts.

The indicators and artifacts devised are counted for each submission (students can submit multiple times) and tagged with the student id and order of program to create a matrix of programs. Programs that have similar variable counts are defined as more similar. This method of defining similarity provides direct connections to concepts in CT and aids in interpreting states and paths found in the subsequent steps.

**Table 1.** Operationalization of Features

| Feature Focus | Programming Aspects Counted |
|---|---|
| Pattern recognition | Board checks, list functions |
| Abstraction | Board checks, list functions, Boolean logic |
| Decomposition | Student defined functions, regular functions (actions) |
| Algorithms | Control flow statements |
| Effort and Efficiency | Lines |

The analysis of student paths is the next step, performed through the creation of a hidden Markov model of the entire group. In our case, we are aiming to chart the progress of a student through various milestones, our hidden states, by observing the student's submission similarities to other student code. This was chosen due to its use in previous works and for its simplicity and ease of understanding (Blikstein et al., 2014).

To generate the states of our hidden Markov model, we use k-medoid clustering on the similarity matrix to produce counting instances of CT subskills and lines used. k-medoid clustering groups items such that the median of the group is equidistant from all points (Park & Jun, 2009). This helps account for outliers or skewness in the data. The number of clusters is designed to maximize silhouette score, a measure of cluster distance calculated from the data (Kaufman & Rousseeuw, 2009). The clusters with the highest silhouette scores are used as states for the hidden Markov model. States are therefore based on patterns of CT use due to the similarity measure used in this approach.

With our clusters of milestone states and student submissions, we calculate the transmission (again, the probability that a state moves to another state) and emissions probabilities (the probability that a state produces an observation) using a modified Expectation-Maximization algorithm known as the Baum-Welch algorithm in hmmlearn, a Python project dedicated to the creation of hidden Markov models (Weiss et al., 2024; Rabiner, 1989). We then construct a hidden Markov model of the entire group using these probabilities. This hidden Markov model is used to chart the sequence through hidden states for each student with the Viterbi algorithm (Rabiner, 1989).

Our next step in the procedure is to create clusters of student paths. Here, the distance metric for clustering is the Euclidean distance between sequences. This procedure is based on the path analyses performed by Jiang et al. (2022). To compare sequences, we first use dynamic time warping to align student sequences, since students could complete the project with a varying number of steps. Dynamic time warping is a procedure to align sequences by adding or removing states to minimize Euclidean distance (Berndt & Clifford, 1994; see Müller, 2007 for a more thorough explanation). Finally, we then compute clusters of student paths with Euclidean distance between warped sequences, again using k-medoid clustering and prioritizing silhouette score. These clusters of student paths represent different approaches to solving a problem using CT

similarly, but only in this task. This gives insights into the strategies used when utilizing CT in a new challenge, localized to that challenge.

The clusters in combination with indicators of success, i.e., if a student has succeeded in the assignment, may also indirectly indicate CT ability. We suppose that at the most basic level, students who solve an assignment in a state with fewer lines and lower artifacts of CT use, in general, have higher CT abilities than students who solve that same assignment in a state with higher counts. We assume this because students who use fewer markers clearly understand how to decompose the game and solve every issue with less effort. While this is not universally true, even within our assignment, the population with which this process-based assessment is designed is not expected to solve this perfectly or efficiently; rather, a minimal viable program is expected. More simply, any student who solves an assignment has a higher ability than students who do not solve the assignment. Again, these assumptions must be taken with caution given that they do not track every scenario.

### 3.2. Study Design

To illustrate our approach, we recruited 29 students from after school and weekend code clubs organized by a commercial code club organizer. While this is far too small for any validation of a process-based assessment, we use this sample merely to demonstrate the possibilities of process-based assessments in constructionist K–12 contexts. Before recruitment and study procedures, parents and students received an information sheet detailing their rights and procedures of the study and gave their express written consent. The Social and Societal Ethics Committee reviewed the procedure and materials. These students ranged in programming experience from two months to two years. Student ages ranged from 13–17, with most of this sample being male (17% female).

Students first completed a traditional programming assessment based on work by Grover et al. (2015). This test assesses student proficiency in basic algorithmic structures in programming and CT ability through programming. As a traditional assessment, this test uses essay questions that have a singular correct answer and are graded for correctness, with partial credit for minor errors (Grover et al., 2015). This test was chosen due to its established validity, use in similar age groups, and focus on programmatic concepts in CT. We adapted the test for Python-specific structures by changing the code-agnostic questions to use Python-specific code. This test was graded by the main researcher using the grading guide by Grover et al. (2015).

These same students were then tasked with completing the assignment for this process-based assessment within two hours. As mentioned previously, we assigned a game of tic-tac-toe in Python where two players could play on a 3x3 game board, with the starting code for a game board provided. This game board was a list of lists, with game pieces being 1 for X and 2 for O. Students were not allowed to use any packages, or extra code, but they could use the internet, or ask questions of their teacher or each other. Tic-tac-toe was chosen due to its repeated use in teaching and assessing basic computer science and CT skills in non-expert populations (Parham-Mocello et al., 2022).

All students used an online programming portal known as Dodona (https://dodona.be/en/; Van Petegem et al., 2022). Dodona allows students to program online without installing software. An illustration of the environment can be seen in Figure 2. When used as an assessment tool, teachers can assign tests of correctness by specifying inputs to test each submission and outputs that must be printed on the console for each submission. The tests of correctness use the inputs for a submission, which produces an output, which is compared to the teacher-specified outputs. If the outputs match, then the submission is counted as correct. Feedback on the correctness is also shown to the student. An illustration of what a student sees after submission is shown in Figure 2, with tested inputs shown above the output of the program. Each student attempt is saved with student ID, timestamps, errors, and correctness. We used Dodona as an assessment tool where student code must have been able to print “Winner” for a horizontal win, a vertical win, and a diagonal win. Students had unlimited submissions with which to test their work or try new methods of solving the task. After the two-hour assignment period, we downloaded each submission.

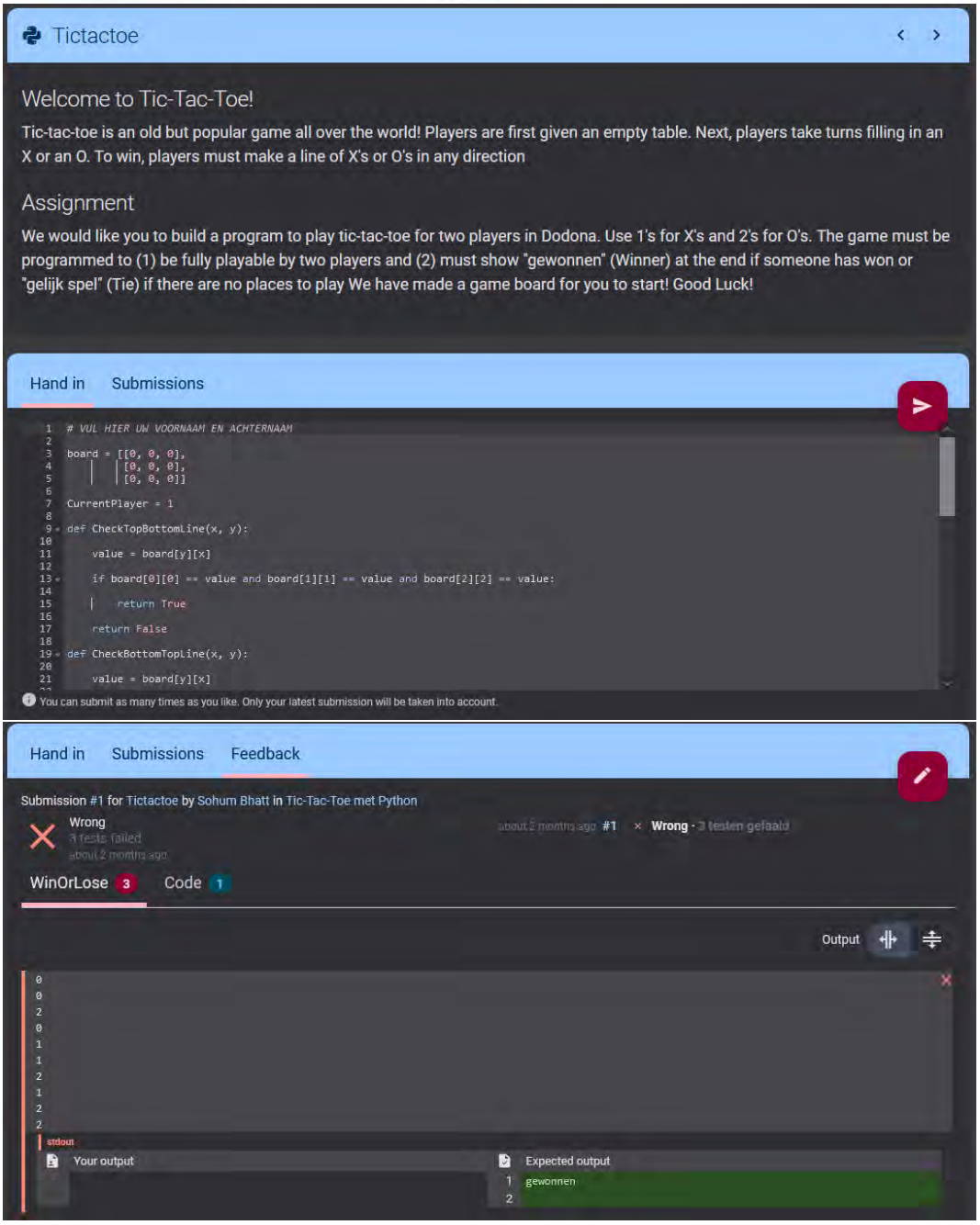


**Figure 2.** Dodona environment and submission test.

To compare our process-based assessment to the traditional assessment, we first used the accumulated student dataset to discover the group hidden Markov model and associated student path clusters. With these clusters, we performed a one-way between-subjects ANOVA to investigate if there are significant differences between student clusters in the grades on the traditional CT exam. This was done to determine if these clusters can be predictive of skill on CT as measured by the traditional exam.

## 4. Results

According to the traditional assessment, students performed relatively well. The average score in the traditional assessment was 14.5 (SD = 7.7) out of a possible 24, indicating that students had a good grasp of programming fundamentals in Python and CT abilities. Three students received perfect scores on this traditional assessment. However two students also chose not to complete the traditional assessment. These two students were left out of further analysis.

The number of submissions on the programming assignment was also high. In total, students tested solutions 346 times, with an average of 10.6 (SD = 8.2) programs submitted per student. Of these submissions, only 14 were marked as correct, meaning that the code ran correctly for each sequence of inputs designed. The average number of lines in each submission was 53.1 (SD = 36.0), comprised of 5.4 (SD = 6.4) board checks, 0.9 (SD = 1.4) list functions, 8.2 (SD = 7.1) instances of Boolean logic, 2.1 (SD = 2.2) user-defined functions, 9.0 (SD = 6.9) regular functions, and 13.4 (SD = 11.7) control flow statements.

### 4.1. Results for Defining States

Four clusters produced the highest silhouette score of 0.56 (a reasonable amount according to Kaufman & Rousseeuw, 2009), suggesting four states for the hidden Markov model. Each cluster successively had more lines, with the first cluster having the least number on average (M = 12.9, SD = 8.1), then the second cluster (M = 48.4, SD = 10.7), the third cluster (M = 87.1, SD = 15.9), and the fourth cluster (M = 111.7, SD = 12.5). The first cluster displayed little usage of all features, whereas the second stage was categorized by relatively even usage of all components used as features. Two other clusters displayed high amounts of specific features, with the third cluster showing the most use of Boolean logic and board checks on average, and the fourth cluster showing the most use of actions, student created functions, and list functions. These four clusters differed in the number of lines and components, and therefore could be interpreted as four distinct stages. The first cluster, categorized by low usage of any feature and low lines was considered the start stage. The second cluster was considered the experimental stage due to the even usage of all components. The third cluster had the highest usage of Boolean logic and was termed the logic stage. Finally, the fourth cluster was focused on functions and actions, which led to its moniker of the "action stage." Differences between all stages can be seen graphically as violin plots of the counts of all features in Figure 3.

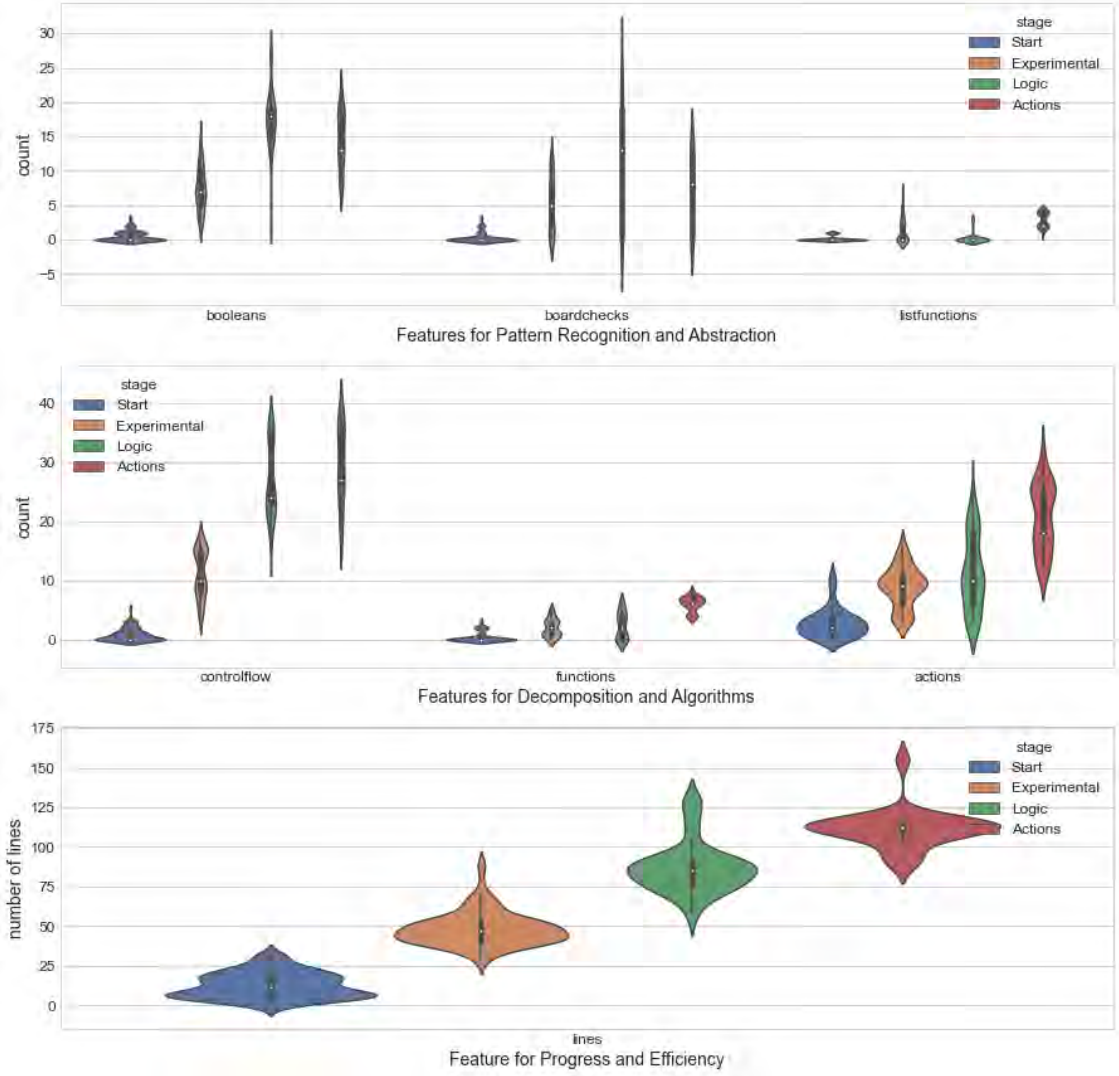


**Figure 3.** Violin plot of features. This plot shows the average amounts for each feature in each stage along with the distribution of the counts of each feature.

### 4.2. Group Hidden Markov Model

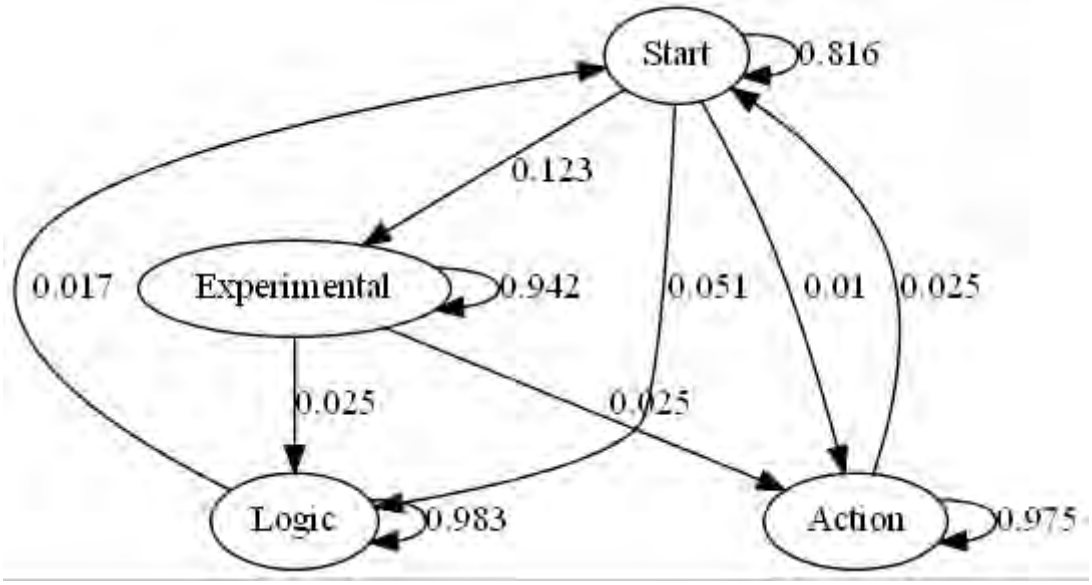


**Figure 4.** Hidden Markov diagram. This diagram displays the probabilities of moving from one hidden state to another.

Our hidden Markov model found that the probability that the initial state of a student was the start phase was .85. The probability of starting in the experimental phase was .07, and the probability of starting in the logic phase was .07. Our hidden Markov model showed that students tend to stay in each stage for many updates before moving to another stage, as in the concept of "sink states" from Berland et al. (2014). The most probable pathway for a student through the stages (without returning to another stage or staying at one stage) was starting in the start stage, then moving to the experimental stage, then continuing to the logic or action stages. However, students could have finished in the experimental stage, logic stage, or action stage. No transition probability was omitted. The entire hidden Markov model is visualized in Figure 4.

### 4.3. Student Path Clusters

A choice of five groups of student paths provided the highest silhouette score of 0.42, which is a weak result according to Kaufman and Rousseeuw (2009). These five groups could be characterized by their most common hidden state. The first path group of seven students, termed the starters, remained in the start stage for every code update. These students, therefore, never completed the exercise. In addition, starters submitted five programs (SD = 5.3) on average. The next path group, termed the experimenters, had seven students. The students in this cluster either started in the experimental stage or moved to the experimental stage within three updates. The experimenters then remained in the experimental stage for most updates, with only one student not finishing in the experimental stage. Five of these seven students completed the exercise. These students submitted 15.9 programs (SD = 8.1) on average. Another path group of 10 students, termed the logic users, varied in patterns of use, but the commonality is updates in the logic stage. Some logic users progressed quickly to the logic stage and remained there until the final update, whereas others experimented first in the experimental or start stages for many updates before moving to the logic stage. Of these 10 students, four completed the assignment, all in the logic stage. These students submitted 11.1 programs (SD = 7.0) on average. Finally, two path groups of students had a high proportion of updates in the action stage. The first group of two students, termed pure action users, moved from the start stage to the experimental stage, then moved to the action stage, where they spent most updates. Neither of these students completed the assignment but submitted 25.5 programs (SD = 6.3) on average. The second path group of one student, termed the balanced action user, moved from the start stage to the experimental stage, then to the action stage. The number of updates in the experimental stage was almost equal to the number of updates in the action stage. This student completed the assignment with the action stage in 38 programs.

### 4.4. Interpretation of Paths

The paths produced follow patterns found in previous process-based assessments, even with the lower sample size. As such, the characteristic differences found between the path groups may have implications for differences in CT ability. At the most basic level, the starters group may have had low CT ability, as they failed to even experiment with possible solutions and no student completed the assignment. However, given the low number of submitted programs, they were more likely to be

disengaged or demotivated. Experimenters, as a group, submitted five more programs than the average for the entire sample but had the highest proportion of students complete the task successfully. When combined with the knowledge that most students remained in the experimental stage for most program submissions, this indicates that students were balanced in all CT skills and used a variety of techniques to solve the challenge. These students likely have the highest real-world CT ability. The logic users and the pure action users were similar in their paths, with the main difference being the focus on logic and functions, respectively. This difference, along with the low proportion of students who completed the task in either group, indicates that these students had weaker abilities in real-world CT scenarios, but these weaknesses stemmed from different subskills. The logic users show reliance on Boolean logic and fail to use actions and functions effectively; compared to the experimenters, logic users have high amounts of board checks and uses of Boolean logic. This indicates brute force attempts to catch patterns needed to solve the tic-tac-toe game. Similarly, the higher amounts of student-designed functions, actions, and special functions used by the pure action group indicate inefficient uses of board checks and Boolean logic. In contrast, the balanced action user has higher ability than pure action users in general. This can be seen in the balance between experimental stages and action stages and the completion of the assignment. However, given that only one balanced action user was captured, it is difficult to compare balanced action and experimenter groups. Despite these path differences, no significant differences were found between the groups in the score on the traditional CT assessment ($F(4, 22) = 0.85$, $p = .51$).

## 5. Discussion

This research investigated process-based assessments for CT in secondary school students. We produced a procedure for developing simple process-based assessments specific to constructionist tasks in K–12 CT contexts. This procedure first asks users to link aspects of code to CT subskills, then use these connections to define similarity to develop a hidden Markov model. This hidden Markov model can then be used to subdivide students based on their paths through the assignment. We then demonstrated this procedure with a tic-tac-toe task completed by a sample of 29 students. With this sample, we aimed to answer two research questions:

1. What CT processes can we observe during a project task?
2. How do the results of a process-based assessment compare to a traditional assessment?

For research question 1, we found that students tend to program in 4 phases: a start phase, an experimental phase, a logic phase, and an action phase. Students remained in these phases for many updates before moving to another phase. The most probable path for students was starting in the start phase, then moving to the experimental phase, before moving to the logic or action stages. Students could also be grouped into five different groups based on how they worked through the assignment. For research question 2, these groups did not differ in ability measured by a traditional assessment, but differences in the amount of completion and the most used stages in a path may have some relation to real-world CT ability. We now discuss the creation of process-based assessments using our procedure, how these pathways and groups of pathways found in CT process-based assessments could be used as a formative assessment, compare our work to other such works, and discuss the viability of process-based assessments as summative assessments in K–12 contexts.

Our procedure for developing process-based assessment produces specific feedback on the task assigned. The specific feedback, by design, stems from constructionist theory. Constructionist theory specifies the importance of personal, meaningful experiences as a guiding force behind learning. As such, feedback must be specific to the project that a student has completed, such as the critiques discussed earlier. While this limits the generalizability of process-based assessments produced by our procedure, we believe that developing tools to help teachers and developers create process-based assessments provides more value than a single process-based assessment by allowing teachers to personalize them more actively. With this procedure, to develop a process-based assessment for other tasks, developers, teachers, or researchers must first break down the task into problems that must be solved with CT. These actions must then be connected to CT concepts. Again, we stress that this step is not intended to only capture actions needed to produce an outcome, but to discretize how CT can be used during the assignment. This stage is the most important in developing a process-based assessment with our procedure, and following this step, our procedure can be followed completely.

Process-based assessments built using our methodology could provide a new avenue for formative assessment in CT. First, according to previous research, process-based assessments can act as a demonstration of quality and provide avenues to begin comparing different approaches, such as in Figure 4 (Grover, 2021). Second, process-based assessments can provide process feedback. Previous research in constructionist theory and CT advocates for the use of process feedback for learning tasks, specifically to provide direct insight into the nature of student misconceptions and errors, rather than relying on indirect questioning through traditional assessments (Berland et al., 2014; Grover, 2021). This insight can be used to provide process feedback to students, directly instructing students on how to improve based on an individual student's code. Beyond an individual level, according to Sedrakyan et al. (2016), groups of paths found in process-based assessments can be used to help provide specific evidence to students to improve cognitive processes in future assignments. Process feedback in general can contribute to guidance that is currently lacking in CT instruction and contribute to stronger constructionist education

(Berland et al., 2014; Lye & Koh, 2014). In practice, we foresee two methods that could allow process-based assessments built using our methodology to be implemented for ease of use: first, process-based assessments could be integrated into a learning tool that automatically creates a Markov model for each class; second, a model could be built for an entire region, and a learning tool would automatically categorize students into a subgroup from that model. However, these ideas need much more elaboration and future work. Future work in CT process-based assessments can also investigate how teachers interpret the results of process-based assessments and where teachers focus their feedback.

In comparison to previous work, despite the low number of students in our sample, the path of our group is like those found by Berland et al. (2013) and Blikstein et al. (2014). Like Berland et al. (2013), we found clear evidence of tinkering, evidenced by many students entering and repeating the experimental stage. However, we found less evidence of distinct phases of tinkering (Berland et al., 2013). This difference might stem from sample differences, as our sample had programming experience, unlike Berland et al. (2013). Therefore, we may have seen tinkering without language exploration. In comparison to the state machine from Blikstein et al. (2014), we found that all the stages in our group path could be considered "sink states," or a state where students remain for many updates. "Sink states" in their initial interpretation were states with difficult bugs or problems, however, in our work, sink states are interpreted as different strategies toward solving the same problem (Blikstein et al., 2014). The two interpretations of these sink states may affect implications on ability, so more study or a better understanding of the origin of these sink states is vital. It is also possible that both interpretations of sink states are valid, depending on the definition of similarity (e.g., AST similarity or primitives). Regardless, informing students of difficult states or different approaches to solving the same problem could help them better utilize metacognitive processes to accomplish the task required. It is possible that some of the differences in our work were due to the definition of similarity, which is the basis for interpreting results in process-based assessments. As such, future work in process-based assessments should be explicit in how they are connecting CT constructs to artifacts found during the development of a project, and for what theoretical or practical reason. In addition, future research should compare the influence of different similarity measures on interpretation and agreement of student paths in the same dataset.

When compared to groups of paths found in other studies, our groups of students did not differ in ability when measured on a traditional assessment (Blikstein et al., 2014). This may be due to random fluctuation (given the small number of participants in our study), but also to the theoretical basis of this work or assessments used. Constructionist works often do not directly relate to test scores or other cognitive metrics, as the focus is on developing, understanding, applying, and assessing understanding of the underlying principles used for a project rather than rote memorization or specific steps (Berland et al., 2014). As such, our assessment may have been focused on other constructs not traditionally assessed. In addition, traditional assessments in programming and computer science are focused on syntactical understanding and code writing skill. These assessments may be more closely related to the specific focus on programming ability as in previous process-based assessments due to the use of features that more closely relate to syntax.

As mentioned previously, it was unknown whether process-based assessments could be used for individual classes in K–12 education. Along with previous research that process-based assessments can be used as formative assessments, we aim to demonstrate such an example with our work (Berland et al., 2014). However, for summative assessment, while our intention is not to use process-based assessment as summative assessment, our study finds that the stability of student groups found in a process-based assessment is unknown. In our study, the number of clusters for the number of students was relatively high given the number of participants, indicating that our clusters may not be stable. This means that, in another small sample or even another randomization of starting conditions, we may see more, fewer, or different clusters of students. Future research should replicate this work on a larger sample size to see if these same clusters exist and can be interpreted similarly. More fundamental to the use of process-based assessments as summative assessments, it is unknown whether new students can be easily categorized into these groups in future assignments, which also requires more research. These aspects of stability may be important for grading and evaluation, which is the final goal of summative assessments.

### 5.1. Limitations

One limitation of our approach is that it is designed specifically for constructionist programming tasks in CT. In these contexts, closely linked with computer science, assignments and activities are intentionally structured to have similar endpoints that can be compared, with many choices within that space (Silapachote & Srisuphab, 2016). In scenarios with more open-ended projects, like mathematics, it is unknown how well the procedure can generate insights and allow for feedback. Here, we identify a need for future research, as the procedure of creating a process-based assessment is even more valuable, allowing researchers and teachers to account for the task-specific nature of CT and use constructionism in contexts beyond programming. Our approach can be further investigated or serve as inspiration for new approaches. Yet, it is likely that any constructionist approach for CT in specific domains will rely on making connections between CT and the task at hand. However, this step is the most difficult for researchers, and therefore likely for teachers and developers as well. This difficulty has been discussed before when applying data-driven approaches to constructionist assessments, however, for CT there is added difficulty (Berland et al., 2014). Future investigations should specifically address how this difficulty can

be addressed, such as what patterns are recognized by experts versus novices. Future research can also investigate differences in the development of process-based assessments between expert and novice teachers or developers.

Another limitation is that it is difficult to evaluate the criterion validity of our approach. Traditional assessments in CT have diverse focuses in assessing CT from other abilities, as in the case of the traditional assessment we adapted from Grover et al. (2015), or in directly assessing problem-solving ability (Tang et al., 2020; Veerasamy et al., 2016). In addition, each assessment may have a different definition of CT (Tang et al., 2020). As such, it is unclear whether a strong association between traditional and process-based assessments in CT is even desirable. In addition, our use of a sample that is experienced with programming may have made it difficult to see differences in CT using the traditional assessment adapted from Grover et al. (2015). Regardless, a further validation of the approach should be addressed in future work.

The last limitation is our small sample size, meaning our results could be affected by random variation within and across subjects. However, our results have some level of agreement with previous work in process-based assessments, giving confidence that we have measured underlying phenomena rather than only random variation. The small sample size also made it difficult to compare groups of paths and assess real-world CT competence, e.g., the pure action and balanced action users. In addition, we may not have been able to model more fine-grained stages or transitions seen in previous works. More research is required to see if this is a consequence of the sample size, the assignment used, or the particular K–12 context. However, the focus of our work was not on validating a process-based assessment, but rather presenting a procedure to broadly develop process-based assessments that can be used in constructionist tasks for K–12 CT education. With this constructionist lens, the focus of process-based assessment is on providing formative feedback. With such a formative use context, teachers are recommended to tailor formative assessments to the interests and needs of their students (Grover, 2021; Lye & Koh, 2014). Following these recommendations and our procedure, teachers using process-based assessments in K–12 CT classes may find it difficult to see precise step information as shown in Blikstein et al. (2014) or Berland et al. (2013).

## 6. Conclusion

This research proposes a procedure for developing process-based assessments for CT using constructionist projects. We also demonstrate this procedure with a constructionist task in a small sample of students. We show that process-based assessments can show distinct phases of CT use and group students into distinct patterns of use that may allow for process feedback. However these phases and groups are based on the assignment given. Therefore, skill assessment for process-based assessments in CT may be dependent on the task given. We also find that process-based assessments produced with our procedure may not relate to traditional assessment, unlike prior literature. However, given that this also occurs with constructionist tasks and that also traditional assessments in CT have limitations, it is unclear whether a strong association is desired.

## Declaration of Conflicting Interest

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The publication of this article received financial support from the Flanders Agency of Innovation & Entrepreneurship under grant number AH.2019.05.

## Acknowledgements

We would like to acknowledge Christine Zhao for her contributions to this work by discussing and investigating CT as part of her Master's thesis. We would also like to thank FTRPRF for their aid in recruiting subjects for this work.

## References

Almeida, F., & Xexéo, G. (2019). *Word embeddings: A survey*. arXiv:1901.09069. https://doi.org/10.48550/arXiv.1901.09069

Angeli, C. (2022). The effects of scaffolded programming scripts on pre-service teachers' computational thinking: Developing algorithmic thinking through programming robots. *International Journal of Child–Computer Interaction*, *31*, 100329. https://doi.org/10.1016/j.ijcci.2021.100329

Beheshti, E. (2017). Computational thinking (CT) in practice: How STEM professionals use CT in their work. *Proceedings of the American Educational Research Association Annual Conference* (AERA 2017), 27 April–1 May 2017, San Antonio, TX, USA. https://par.nsf.gov/biblio/10026245-computational-thinking-practice-how-stem-professionals-use-ct-work

Bennett, R. E. (2011). Formative assessment: A critical review. *Assessment in Education*, *18*(1), 5–25. https://doi.org/10.1080/0969594X.2010.513678

Berland, M., Baker, R. S., & Blikstein, P. (2014). Educational data mining and learning analytics: Applications to constructionist research. *Technology, Knowledge and Learning*, *19*(1), 205–220. https://doi.org/10.1007/s10758-014-9223-7

Berland, M., Martin, T., Benton, T., Smith, C. P., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, *22*(4), 564–599. https://doi.org/10.1080/10508406.2013.836655

Berndt, D. J., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (AAAIWS '94), 31 July–1 August 1994, Seattle, WA, USA (pp. 359–370). AAAI Press. https://dl.acm.org/doi/10.5555/3000850.3000887

Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. *Proceedings of the 1st International Conference on Learning Analytics and Knowledge* (LAK '11), 27 February–1 March 2011, Banff, AB, Canada (pp. 110–116). ACM Press. https://doi.org/10.1145/2090116.2090132

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, *23*(4), 561–599. https://doi.org/10.1080/10508406.2014.954750

Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M., Jasutė, E., Malagoli, C., Masiulionytė-Dagienė, V., & Stupurienė, G. (2022). *Reviewing computational thinking in compulsory education: State of play and practices from computing education* (A. Inamorato dos Santos, R. Cachia, N, Giannoutsou, & Y. Punie, Eds.). Publications Office of the European Union. https://doi.org/10.2760/126955

Bonner, S., Chen, P., Jones, K., & Milonovich, B. (2021). Formative assessment of computational thinking: Cognitive and metacognitive processes. *Applied Measurement in Education*, *34*(1), 27–45. https://doi.org/10.1080/08957347.2020.1835912

Boom, K.-D., Bower, M., Siemon, J., & Arguel, A. (2022). Relationships between computational thinking and the quality of computer programs. *Education and Information Technologies*, *27*(6), 8289–8310. https://doi.org/10.1007/s10639-022-10921-z

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the American Educational Research Association Annual Conference* (AERA 2012), 13–17 April 2012, Vancouver, BC, Canada.

Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., Joshi, D., Robinson, R., & Andrews, A. (2019). PRADA: A practical model for integrating computational thinking in K–12 education. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (SIGCSE '19), 27 February–2 March 2019, Minneapolis, MN, USA (pp. 906–912). ACM Press. https://doi.org/10.1145/3287324.3287431

Fields, D. A., Lui, D., & Kafai, Y. B. (2019). Teaching computational thinking with electronic textiles: Modeling iterative practices and supporting personal projects in *exploring computer science*. In S.-C. Kong & H. Abelson (Eds.), *Computational thinking education* (pp. 279–294). Springer, Singapore. https://doi.org/10.1007/978-981-13-6528-7_16

Grover, S. (2021). Toward a framework for formative assessment of conceptual learning in K–12 computer science classrooms. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (SIGCSE '21), 13–20 March 2021, Virtual, USA (pp. 31–37). ACM Press. https://doi.org/10.1145/3408877.3432460

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Hartmann, C., Rummel, N., & Bannert, M. (2022). Using HeuristicsMiner to analyze problem-solving processes: Exemplary use case of a productive-failure study. *Journal of Learning Analytics*, *9*(2), 66–86. https://doi.org/10.18608/jla.2022.7363

Jiang, B., Zhao, W., Zhang, N., & Qiu, F. (2022). Programming trajectories analytics in block-based programming language learning. *Interactive Learning Environments*, *30*(1), 113–126. https://doi.org/10.1080/10494820.2019.1643741

Kaufman, L., & Rousseeuw, P. J. (2009). *Finding groups in data: An introduction to cluster analysis*. Wiley.

Kite, V., & Park, S. (2023). Context matters: Secondary science teachers' integration of process-based, unplugged computational thinking into science curriculum. *Journal of Research in Science Teaching*, *61*(1), 203–227. https://doi.org/10.1002/tea.21883

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child–Computer Interaction*, *2*(1), 26–33. https://doi.org/10.1016/j.ijcci.2014.06.003

Liu, Q., & Paquette, L. (2023). Using submission log data to investigate novice programmers' employment of debugging strategies. *Proceedings of the 13th International Conference on Learning Analytics and Knowledge* (LAK '23), 13–17 March 2023, Arlington, TX, USA (pp. 637–643). ACM Press. https://doi.org/10.1145/3576050.3576094

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, *30*(4), 883–908. https://doi.org/10.1007/s11191-021-00202-5

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*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Ma, J., Zhang, Y., Bin, H., Wang, K., Liu, J., & Gao, H. (2022). The development of students' computational thinking practices in AI course using the game-based learning: A case study. *2022 International Symposium on Educational Technology* (ISET), 19–22 July 2022, Hong Kong, China (pp. 273–277). https://doi.org/10.1109/ISET55194.2022.00065

Müller, M. (2007). Dynamic time warping. In *Information retrieval for music and motion* (pp. 69–84). Springer. https://doi.org/10.1007/978-3-540-74048-3_4

Neamtiu, I., Foster, J. S., & Hicks, M. (2005). Understanding source code evolution using abstract syntax tree matching. *Proceedings of the 2005 International Workshop on Mining Software Repositories* (MSR '05), 17 May 2005, St. Louis, MO, USA (pp. 1–5). ACM Press. https://doi.org/10.1145/1083142.1083143

National Research Council. (2013). *Next generation science standards: For states, by states*. The National Academies Press. https://doi.org/10.17226/18290

OECD. (2020). *PISA 2018 results: Effective policies, successful schools* (Vol. V). OECD Publishing. https://doi.org/10.1787/ca768d40-en

Oliver, K. M., Houchins, J. K., Moore, R. L., & Wang, C. (2021). Informing makerspace outcomes through a linguistic analysis of written and video-recorded project assessments. *International Journal of Science and Mathematics Education*, *19*(2), 333–354. https://doi.org/10.1007/s10763-020-10060-2

Papavlasopoulou, S., Giannakos, M. N., & Jaccheri, L. (2019). Exploring children's learning experience in constructionism-based coding activities through design-based research. *Computers in Human Behavior*, *99*, 415–427. https://doi.org/10.1016/j.chb.2019.01.008

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Parham-Mocello, J., Nelson, A., & Erwig, M. (2022). Exploring the use of games and a domain-specific teaching language in CS0. *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE '22), 8–13 July 2022, Dublin, Ireland (pp. 351–357). ACM Press. https://doi.org/10.1145/3502718.3524812

Park, H.-S., & Jun, C.-H. (2009). A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, *36*(2, Part 2), 3336–3341. https://doi.org/10.1016/j.eswa.2008.01.039

Piech, C., Sahami, M., Huang, J., & Guibas, L. (2015). Autonomously generating hints by inferring problem solving policies. *Proceedings of the 2nd ACM Conference on Learning @ Scale* (L@S 2015), 14–18 March 2015, Vancouver, BC, Canada (pp. 195–204). ACM Press. https://doi.org/10.1145/2724660.2724668

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286. https://doi.org/10.1109/5.18626

Rich, P. J., Egan, G., & Ellsworth, J. (2019). A framework for decomposition in computational thinking. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE '19), 15–17 July 2015, Aberdeen, Scotland, UK (pp. 416–421). ACM Press. https://doi.org/10.1145/3304221.3319793

Roque, R., & Tamashiro, M. A. (2022). Making learning visible in constructionist learning contexts. *Proceedings of the 21st Annual ACM Interaction Design and Children Conference* (IDC '22), 27–30 June 2022, Braga, Portugal (pp. 69–81). ACM Press. https://doi.org/10.1145/3501712.3534093

Rosé, C. P., McLaughlin, E. A., Liu, R., & Koedinger, K. R. (2019). Explanatory learner models: Why machine learning (alone) is not the answer. *British Journal of Educational Technology*, *50*(6), 2943–2958. https://doi.org/10.1111/bjet.12858

Sands, P., Yadav, A., & Good, J. (2018). Computational thinking in K–12: In-service teacher perceptions of computational thinking. In M. S. Khine (Ed.), *Computational thinking in the STEM disciplines: Foundations and research highlights* (pp. 151–164). Springer, Cham. https://doi.org/10.1007/978-3-319-93566-9_8

Sedrakyan, G., De Weerdt, J., & Snoeck, M. (2016). Process-mining enabled feedback: "Tell me *what* I did wrong" vs. "tell me *how* to do it right." *Computers in Human Behavior*, *57*, 352–376. https://doi.org/10.1016/j.chb.2015.12.040

Silapachote, P., & Srisuphab, A. (2016). Teaching and learning computational thinking through solving problems in Artificial Intelligence: On designing introductory engineering and computing courses. *2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering* (TALE), 7–9 December 2016, Bangkok, Thailand (pp. 50–54). https://doi.org/10.1109/TALE.2016.7851769

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798. https://doi.org/10.1016/j.compedu.2019.103798

Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K–12 education: A conceptual model based on a systematic literature review. *Computers & Education*, *162*, 104083. https://doi.org/10.1016/j.compedu.2020.104083

Van Petegem, C., Deconinck, L., Mourisse, D., Maertens, R., Strijbol, N., Dhoedt, B., De Wever, B., Dawyndt, P., & Mesuere, B. (2022). Pass/fail prediction in programming courses. *Journal of Educational Computing Research*, *61*(1), 68–95. https://doi.org/10.1177/07356331221085595

Veerasamy, A. K., D'Souza, D., & Laakso, M.-J. (2016). Identifying novice student programming misconceptions and errors from summative assessments. *Journal of Educational Technology Systems*, *45*(1), 50–73. https://doi.org/10.1177/0047239515627263

Weintrop, D., Wise Rutstein, D., Bienkowski, M., & McGee, S. (2021). Assessing computational thinking: An overview of the field. *Computer Science Education*, *31*(2), 113–116. https://doi.org/10.1080/08993408.2021.1918380

Weiss, R., Du, S., Grobler, J., Cournapeau, D., Pedregpsa, F., Varoquaux, G., Mueller, A., Thirion, B., Nouri, D., Louppe, G., Vanderplas, J., Benediktsson, J., Buitinck, L., Korobov, M., McGibbon, R., Lattarini, S., Niculae, V., csytracy, Gramfort, A., … Rockhill, A. (2024). *hmmlearn* (Version 0.3.2) [Data set]. GitHub. https://github.com/hmmlearn/hmmlearn

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wong, G. K. W., & Jiang, S. (2018). Computational thinking education for children: Algorithmic thinking and debugging. *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering* (TALE), 4–7 December 2018, Wollongong, NSW, Australia (pp. 328–334). https://doi.org/10.1109/TALE.2018.8615232