Computational Thinking as a Practice of Representation: A Proposed Learning and Assessment Framework

Camilo Vieira cvieira@purdue.edu
Manoj Penmetcha mpenmetc@purdue.edu
Alejandra J. Magana admagana@purdue.edu
Eric Matson ematson@purdue.edu

Computer and Information Technology, Purdue University,
401 N. Grant Street, West Lafayette, IN. 47906

ABSTRACT

This study proposes a research and learning framework for developing and assessing computational thinking under the lens of representational fluency. Representational fluency refers to individuals' ability to (a) comprehend the equivalence of different modes of representation and (b) make transformations from one representation to another. Representational fluency was used in this study to guide the design of a robotics lab. This lab experience consisted of a multiple step process in which students were provided with a learning strategy so they could familiarize themselves with representational techniques for algorithm design and the robot programming language. The guiding research question for this exploratory study was: Can we design a learning experience to effectively support individuals' computing representational fluency? We employed representational fluency as a framework for the design of computing learning experiences as well as for the investigation of student computational thinking. Findings from the implementation of this framework to the design of robotics tasks suggest that the learning experiences might have helped students increase their computing representational fluency. Moreover, several participants identified that the robotics activities were engaging and that the activities also increased their interest both in algorithm design and robotics. Implications of these findings relate to the use of representational fluency coupled with robotics to integrate computing skills in diverse disciplines.

Categories and Subject Descriptors

K.3.2 [Computers And Education]: Computer and Information Science Education – *Computer science*

General Terms

Algorithms, Human Factors

Keywords

Computation, Representational Fluency, Programming Education, Robotics

1. INTRODUCTION

Calls for action in the field of computer science education and computing educational research have identified, among other issues, the lack of a variety of methodological approaches to the design and investigation of computing learning experiences [i.e. 1, 2, 3]. These calls for action are based on searches of published research literature in which authors have concluded that there is a relative sparseness of research regarding how students learn computer science and, a lack of rigor in most of the existing investigations [2]. As a pathway to addressing this need, Clement [1] proposed applying findings from science education to the design of evidence-based learning experiences in computer science. We would like to extend this call and include the use of theoretical frameworks in the evaluation of student learning and not only in the design stage.

Our aim is first and foremost to contribute to the field of computing education by proposing the use of representational fluency as a theoretical framework for the design of computing learning experiences as well as a way to investigate how students learn computer science related concepts under this lens. To this end, the guiding research question is: Can we design learning experiences to effectively support individuals' computing representational fluency? Specifically, this study proposes a learning experience that uses representational fluency as a way for students to develop computational thinking mediated by the use of robotics. The research questions that helped us assess this proposed approach are:

(i) What are individuals' representational abilities for problem solving in the context of robotics challenges?

(ii) What is the effect of computational robotics challenges for improving individuals' computing representational fluency?

(iii) Do individuals' background (computing or non-computing), academic level (freshmen or sophomore), and/or gender have an effect in their computing representational abilities for problem solving in the context of a robotics problem solving task?

(iv) What are individuals' perceptions about the usefulness of computational robotics challenges to learn algorithmic design and robotics?

We believe that representational fluency can help us (a) to design learning experiences that can help students manage complexity by means of abstractions and (b) have a clearer understanding of how learners learn and develop expertise in computational thinking. Findings can then inform effective methods and pedagogies to train the next generation of workers with readily available computing skills.

2. Background

We begin with a definition of computational thinking and its relationship with abstraction. We then explore some of the learning difficulties in the field of computer science education and briefly describe the role of robotics as a pedagogical and motivational tool to integrate computational thinking in the context of problem solving. Next, we make an argument of how computational thinking relates to representational fluency and proceed to the application of the proposed theoretical framework to the design of a robotics learning activity. Finally, we assess the effectiveness of this approach by means of an exploratory study.

2.1 Computational Thinking

Computational thinking [4] has been recognized as a collection of understandings and skills required for new generations of students to be proficient not only at using tools, but also at creating them and understanding the nature and implication of that creation [5]. Computational thinking refers to the combination of disciplinary knowledge (e.g. physics, biology, nanotechnology) [6] with thought processes (e.g. engineering thinking, quantitative reasoning, algorithmic thinking, systems thinking) involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent [7]. This requires using a set of concepts, such as abstraction, recursion, and iteration, to process and analyze data and to create real and virtual artifacts [8, 9].

The use and creation of computing models are an important step in understanding problems and identifying potential solutions. Algorithmic thinking and abstraction are two of the constructs that are at the core of computational thinking. Algorithmic thinking consists of the ability to perform "functional decomposition, repetition (iteration and/or recursion), basic data organizations (record, array, list), generalization and parameterization, algorithm vs. program, top-down design, and refinement" [10]. Abstraction refers to the act or process of removing detail to simplify and focus attention to salient characteristics based on a given criteria [11]. Therefore, investigations of what it means to solve problems through different forms of representations, in which students need to couple abstraction with algorithmic thinking in the context of computational problem solving tasks, should result in productive venues to advance relevant learning science theories [12, 13].

2.2 Challenges in Computer Science Education

Research described in the computer science education literature has identified for a long time that learning to program is difficult [14-16]. For instance, computer programs, in order to function appropriately, require some level of complexity and adherence to formalisms. Some identified difficulties occur in following areas: (i) orientation- to identify the purpose of the programming task; (ii) the notional machine- to identify the general properties or functionality of the machine that one is intending to control; (iii) notation- to master the syntax and semantics of the programming language; (iv) structure- to deal with the difficulties of acquiring standard patterns or schemas that can be implemented to attain small-scale goals; and (v) pragmatics- to develop the skills to be able to specify, develop, test, and debug programs using whatever tools are available [17, 18]. Consequently, teaching programming to people who are not familiar with algorithm design (at least flow diagrams design) can also be a hard task. The process includes not only abstraction and algorithmic design capabilities, but also programming languages syntax and semantics (Cliburn, 2006). Additionally, the non-user-friendly outcomes of a program might become a constraint leading to lack of motivation on the part of the students.

2.3 Robotics in Computer Science Education

Robotics has been included in computing science classes and curricula as one of the strategies to teach artificial intelligence and programming in an engaging way [19-22]. Several studies using tools such as Lego Mindstorms [19, 23], Robocode [24, 25] or Moway [26] have explored the development of programming skills coupled with robotics. Klassner & Anderson (2003) highlighted its use in areas such as: Programming Fundamentals to learn conditionals, loops, and object-oriented paradigm; Algorithms and Complexity to be aware of efficiency in order to improve battery lifetime and the motion speed; and Programming Languages, Architecture, and Operative Systems to understand concepts such as syntax and multitasking. Cilburn (2006) also highlighted its usefulness for beginner courses, such as Fundamental Concepts of Computer Science, in which students prefer building and programming over lecture courses. On the other hand, he found that in some robotics experiences there might be external factors that frustrate students, such as light sensors that may be affected by the environmental light or battery life.

While the computer science community has taken strides to address issues of methodological rigor in their investigations, to date, little work has been done to apply existing learning theories and theoretical frameworks to the design of learning experiences and also to create new discipline-specific learning theories. This study attempts to use research from the learning sciences to link constructs of initial learning conditions, initial learning context, problem representations, transfer as an active, dynamic process, and, specifically, representational fluency for computational thinking.

3. Theoretical Foundations

Expertise, transfer, and representational fluency are key theoretical constructs that guided the design of the learning experience and subsequent investigation.

3.1 Expertise

Expertise consists of those characteristics, skills, and knowledge of a person (that is, expert) or of a system, which distinguish experts from novices and less experienced people. A sine qua non characteristic of experts is the ability to fluently transfer what they have learned from one situation to another; novices cannot do this. Novices' learning is closely connected to the conditions in which they learn; novices tie principles and concepts that they know to the surface features of how they were taught the principle or concept. Consequently, when the context changes, novices often fail to transfer the principle to a new situation. Experts, on the other hand, have abstracted the knowledge that is associated with a particular context. This abstracted knowledge is based on principles and is usually derived from repeated learning across varying contexts where the need for abstraction is designed into the problem.

Experts possess both general problem solving skills and domain knowledge. Furthermore, there is a symbiotic relationship between general cognitive skills and domain-specific knowledge: "general heuristics that fail to make contact with a rich, domain-specific knowledge base are weak. But when a domain-specific knowledge base operates without general heuristics, it is brittle—it serves mostly in handling formulaic problems" [27]. These are important points to remember as we consider the design, development and evaluation of educational environments that contribute to the development of expertise. Expertise does not magically happen. The development of expertise is a complex phenomenon. One useful perspective for approaching and understanding the design and development of educational environments that contribute to the development of expertise is through exploration of the construct of "transfer." A second perspective underpinning this project is that of representational fluency.

3.2 Transfer

"Transfer" is about educating so that the learner will be able to use the newly acquired knowledge on a different problem, in a different situation, and it is not about simply training people to accomplish tasks ([NRC], 2000). A common goal for educators is to help the learner acquire knowledge that extends to other contexts. In 2000, The National Research Council published findings that suggest the following key characteristics of learning and transfer that are helpful for educators:

- Initial learning is necessary for transfer, and a considerable amount is known about the conditions of initial learning experiences that support transfer.
- All new learning involves transfer based on previous learning. Transfer is affected by the context of initial learning, and this has important implications for the design of instruction that helps students learn.
- Knowledge that is overly contextualized can reduce transfer; transfer is enhanced by instruction that guides students toward the representation of problems at higher levels of abstraction.
- Transfer is best viewed as an active, dynamic process rather than a passive end-product of a particular set of learning experiences.

Conditions of Initial Learning. Initial learning is a key factor for transfer, and it is often overlooked. Initial learning consists of mastery of a particular topic or subject matter [28]. In a study to evaluate the effects of transfer when using the programming language LOGO, it was found that there were no benefits of transfer unless a significant degree of knowledge was gained during the learning process [28]. Additionally, further studies have shown that the following characteristics of initial learning that affect transfer are: a) understanding versus memorizing, b) time to learn, c) beyond "Time on Task," and d) motivation to learn [28]. When learners are only required to memorize facts, they may have difficulty understanding the "why?" and the "how come?" By organizing facts around principles, students will better answer these questions and will start to organize a mental framework that more closely resembles that of experts [28]. Moreoever, it is important to understand the amount of time initial learning takes to move knowledge into long-term memory; for example, to become a chess master, an individual requires around 100,000 hours of playing to reach world class expertise [28]. Much of the time spent on initial learning is used to develop patterns of recognition that can be recalled and applied to new experiences [28]. The different ways time is used is also a key factor to initial learning. Deliberate practice with feedback is considered a more effective use of time than spending time practicing without feedback [28]. Motivation should be considered as one of the most important aspects of initial learning and will help the learner stay on-task and dedicate the time necessary to move knowledge into long-term memory. Varying the degrees of difficulty is one way of helping the learner to stay motivated; however, educators should be careful not to make the learning so difficult that the learner loses interest, or so easy that the learner becomes bored [28]. Each of these characteristics of learning (understanding, time to learn, "Time on Task," and motivation) should be considered when providing instruction because each has been shown as important to initial learning conditions that support later transfer.

Initial Learning Context and Transfer. The context in which learning is initially achieved is important to subsequent transfer. It has been shown that learning is situated in practice and that traditional classroom cultures and environments are not the most effective contexts for student learning. Transfer can be better served through authentic practices or cognitive apprenticeships [29]. These authentic practices might include embedding tasks with familiar activities, pointing to different decompositions, and allowing students to generate their own solution paths [29]. While authentic practices can be useful for creating a rich initial learning opportunity, research has also shown that novices often fail to invoke prior learning when the context changes, resulting in poor or no transfer. This can partly be corrected through additional examples in different contexts like providing additional similar cases, "what if" analysis, and the abstraction of general principles [28].

Problem Representation. Problem representations also affect transfer. Research shows that the more abstracted the knowledge, the more transferrable it is [28]. Learning experiences that help students see how problems relate to principles and how those principles can be applied to other situations promote positive transfer. A study of algebra students that involved word problems using mixtures showed that those students who were shown pictures of mixtures did worse when trying to transfer their learning to new problems than did other students that were shown abstract tabular representations [28]. Studies have also shown that when learners develop multiple representations they are better able to transfer knowledge to new domains with increased flexibility [30].

Active, Dynamic Approaches to Transfer. In the literature, transfer is often treated as static, where it is conceived and operationalized as an outcome of learning. An alternate approach is to treat transfer as a dynamic process that requires learners to actively choose strategies, evaluate those strategies, consider relevant resources, and receive timely and relevant feedback. Experts spontaneously transfer appropriate knowledge without prompting, and, when they get stuck, they are usually capable of self-regulating their learning so as to redirect. In other words, experts use metacognition (thinking about thinking) to support transfer by re-invoking initial learning, learning context, and problem representation strategies. Transfer can be improved by treating it as an active, dynamic process wherein metacognitive strategies are taught to learners within the abstraction/transfer process.

3.3 Representational Fluency

Generally, fluency is the ability to express oneself readily and effortlessly, as well as the ability to move effortlessly between the spoken word and the written word, which are two different representations. A representation in the abstract refers to instances that are equivalent in meaning, but different in mode of expression. While the idea of fluency is often associated with the written and spoken word, researchers have extended work fluency and representations to other disciplines, (e.g. physics, biochemistry, and mathematics). The idea of fluency in these other fields includes the ability to comprehend the equivalence of different modes of representation [31], a phenomenon that has been called "representational fluency." In science, technology, engineering, and mathematics, commonly used modes of representation include verbal vs. mathematical, graphical vs. equational, macroscopic vs. microscopic, physical vs. virtual, etc. Representational fluency is the ability to comprehend equivalence in different modes of expression, to read out information presented in different representations, to transform information from one representation to another, and to learn in one representation and apply that learning to another. Therefore, representational fluency is an important aspect of deep conceptual understanding that has been shown to promote transfer and expertise.

3.4 Computational Thinking as a Practice of Representation

One of the main goals of computational thinking involves individuals' ability to define models in the form of algorithms, data analysis, or visualization techniques [8, 32]. A model can be referred to as a tool that (a) serves as an approximate representation of the real item that is being built and (b) helps individuals to work at a higher level of abstraction by bringing out the big picture and by focusing on different aspects of a model [33]. Thus, abstraction is at the core of algorithmic thinking, which at the same time is one of the principles that is right at the heart of computational thinking; however, abstraction is as hard to teach as it is important [34].

We argue that, for accomplishing a working level of abstraction, techniques such as problem decomposition, pattern recognition, and pattern generalization can be fostered by having students familiarize themselves with diverse forms of representations, create these representations, and translate meaning from one representation to another. Hence, we propose the use of representational fluency as a conceptual framework that can help us to identify and describe different forms of computational representations and their application in the manipulation, construction, interpretation, application, revision, and refinement of models through the process of solving real life problems.

4. Methods

The methods of this study describe how we used the framework of representational fluency to design a robotics learning experience and to explore if students benefited from it. We expected that students would develop representational abilities by using the designed robotics lab experience embedding the as use-modify-create strategy. To this end, we developed a test case study exploring the following guiding research questions:

(i) What are individuals' representational abilities for problem solving in the context of robotics challenges?

(ii) What is the effect of computational robotics challenges for improving individuals' computing representational fluency?

(iii) Do individuals' background (computing or non-computing), academic level (freshmen or sophomore), and/or gender have an effect in their computing representational abilities for problem solving in the context of a robotics problem solving task?

(iv) What are individuals' perceptions about the usefulness of computational robotics challenges to learn algorithmic design and robotics?

4.1 Learning materials to scaffold representational fluency

To guide student learning, a lab experience was created guided by the notion of representational fluency. This lab experience consisted of a multiple step process in which students were provided with a framework so they could familiarize themselves with representational techniques for algorithm design and the robot programming language. This strategy has been described as use-modify-create [35]. This scaffolding strategy consists of a three-stage progression of deeper interactions [36]. The main objective of the lab module was to make the robot travel through predefined paths forming simple shapes. This lab module had the following steps:

Introduction. This section provided the overview of the activity. A scenario was presented in the introduction of the lab module in which a fictional company is assumed to supply unmanned robots to the US military services. This fictional company is looking to hire a software developer to program the robot to travel through different predefined routes. The participant had been assumed as a software developer and will work on the entire lab module.

[Use] In this part of the lab module, participants were provided with the process required to make the robot travel through the square path and the programming basics. Participants were provided with a sample of a program. To program the robot, participants need to understand the basic functionalities. For instance, students should know that all the four wheels need to be programmed accordingly. Also, students were presented with the variables and the functions to be used. Specifically, the robot is programmed on two variables (time and speed) and it had four basic functions available to students (i.e. stop, forward, turn right, and turn left). A flowchart and a table were also provided to the participants with explanations concerning the procedure used in programming a robot to make a square path (see Figure 1).

This part of the lab module also provided a manual to assemble a robot. This part of the lab was optional to the user. Setting up RoboPlus software [37] and how to connect Robot to the computer were also explained. RoboPlus is a computer program that consists of instructions to control the robot's actions. After writing the program, the file is saved in .tsk format, which was uploaded into CM 510 (Servo Controller) using RoboPlus software. Figure 2 shows a screenshot of the program's interface.

[Modify] Participants modified the above program to create a program where the robot travels through a rectangle-shaped path. The steps participants followed were: (1) create the pseudo code and flowchart of the path, (2) program the robot, (3) test the robot, (4) assess the accuracy of the program versus the design, and (5) modify your code as necessary.

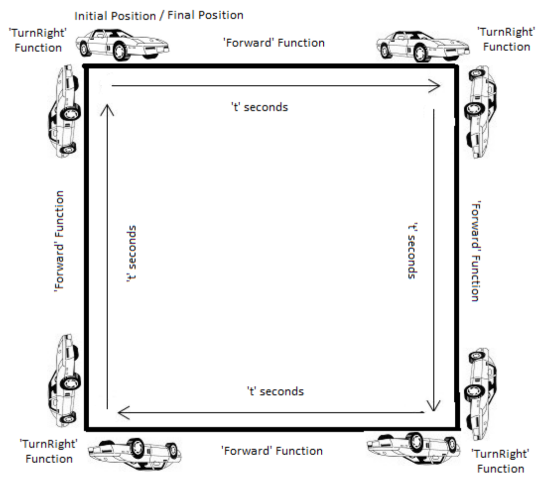
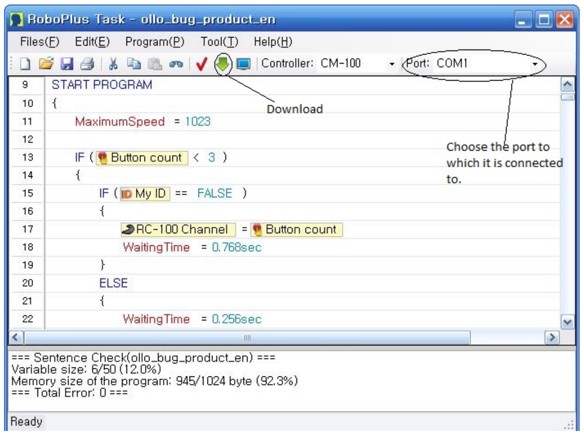


Fig.1 Path of robot making a squared shape.


RoboPlus Task - ollo_bug_product_en
Files(F) Edit(E) Program(P) Tool(T) Help(H)
Download
Choose the port to which it is connected to.
Ready

Controller: CM-100
Port: COM1

```
START PROGRAM
{
    MaximumSpeed  = 1023

    IF ( Button count  <  3 )
    {
        IF ( My ID  ==  FALSE  )
        {
            RC-100 Channel  = Button count
            WaitingTime  = 0.768sec
        }
        ELSE
        {
            WaitingTime  = 0.256sec
```

```
=== Sentence Check(ollo_bug_product_en) ===
Variable size: 6/50 (12.0%)
Memory size of the program: 945/1024 byte (92.3%)
=== Total Error: 0 ===
```

Fig.2 Screenshot of the RoboPlus interface

[Create] After participants became familiarized with the basic concepts of flowchart and programming, they started designing, implementing and testing the robot to accomplish the task assigned and to make the robot travel through pre-defined paths. To this end, participants were guided through a step-by-step scaffolded procedure in which they created diverse forms of representations by building form one to another. The step-by-step procedure was:

Analysis Task. Drawing a flowchart and writing a pseudo code are two forms of representation that participants were asked to perform as part of this task. Each participant was exposed to a natural scenario where he or she was treated as a software developer. The participant was responsible for drawing a flowchart based on the scenario provided (i.e. converting natural language to flowchart).

Design Task. The flowchart produced as part of the Analysis Task was intended to serve as a starting point to then construct the corresponding pseudo code. To create the pseudo code, participants were required to use short English phrases to explain specific instructions needed for the robot to travel the predefined path.

Implementation Task. After creating a flowchart from natural language and then the pseudo code based on the flowchart, participants used those artifacts to program the robot.

Testing Task. The reason for testing was to see if the path traveled by the robot matched the predefined figure. The path traveled by the robot was supposed to be directly related to the program and the deviations from the pre-defined path would indicate the mistakes made in the flowchart, pseudo code, or computer program.

4.2 Participants

Participants of this test case consisted of 44 college students from a Midwestern university with computing (n=16) and non-computing (n=28) backgrounds. The participants in this study were either in their freshmen (n=11) or sophomore (n=33) years. Student majors or disciplines were: Mechanical Engineering (7), Chemistry or Chemical Engineering (5), Computer Engineering or Computer Science (4), Behavioral Neuroscience or Psychology (4), Medical Laboratory Science, Nursing, Health or Applied Exercise (3), Electrical Engineering (2), Biology (2), Biomedical Engineering (2), Business Management (2), Communication (2), Interdisciplinary Engineering (2), Animal/Soil and Crop Science (2), Speech and Language (1), Acting (1), Materials Science and Engineering(1), Aviation Engineering Technology (1), Fine Arts (1), Physics (1), and History (1).

Recruitment of participants was conducted by posting flyers throughout campus. After the participants made initial contact with us, we used a purposeful sampling method. We gave preference to freshmen students. We also gave preference to students from diverse backgrounds (i.e. from a variety of disciplines) in an effort to have a balance between students from computing and non-computing oriented disciplines. Students were then invited to participate in a two hour lab session. This study was approved by the institutional review board.

4.3 Data Collection Method and Procedures

A process assessment rubric (PAR) was employed to evaluate student performance in the planning of the task, implementation of the task, and the program produced. For each step in the process, students were evaluated on the representations they produced and how they translated from one representation to another one; therefore, alignment between representations was considered as part of the rubric to identify how students built from one representation to the following one.

Students' perceptions were collected using three Likert-scale questions scored from strongly disagree (1) to strongly agree (5). The statements to be rated were: (1) The activities presented were very engaging; (2) The activities increased my interest in algorithm design; and (3) The activities increased my interest in robotics.

During the two-hour lab session, students were exposed to three main activities. First, they responded the pretest assessment, then they were exposed to the learning experience, and, finally, they responded the posttest. The perception questions were responded to by the participants at the same time as the posttest.

4.4 Data Analysis Method

All the participants responded to the same pretest and posttest instrument to determine the effects of the treatment on Analysis, Design (flowchart and pseudo-code), and the representational fluency of students among the several artifacts required on the tests (i.e. how they built and aligned the flowchart, pseudo-code, and implementation code). The Implementation score assessed the actual program that manipulated the robot. This category was only scored as part of the posttest assessment. All data from the two rubrics were rated on a scale from 1 to 4, and it was treated as interval data. The responses to the perception questions were normalized so the results ranged from 0% (strongly disagree) to 100% (strongly agree).

All pre and posttest results were tested for normality, none of which were normally distributed. After scoring each rubric individually for the pretest and posttest measures, a non-parametric t-test was used to identify significant differences between the two groups.

A correlational analysis was carried out among the rubric criteria for the pretest and for the posttest. The Pearson coefficient for a weak correlation was considered to be less than 0.1, for a moderate correlation to be between 0.25 and 0.45, and for a strong correlation to be higher than 0.5 [46].

Table 1. Process assessment rubric (PrT=pretest scores, PoT=posttest scores)

Category	4	3	2	1	PrT	PoT
Flowchart Independent to Robotics	All the components are clearly defined, shaped, and labeled. The flowchart describes the process in an accurate manner	The flow chart describes the process, but its components are not correctly labeled, shaped, or defined	Most of the shapes in the flowchart are incorrectly labeled or shaped	The flowchart is incomplete or non-understandable		
Analysis Flowchart	The flowchart design is accurate. Also, it has all the components labeled and shaped. The initial and end steps are clearly represented	The flowchart design is accurate but there are some components that are not correctly labeled, shaped, or defined	The flowchart design lacks of precision to the chosen route and some of the shapes in the flowchart are incorrectly labeled or shaped	The flowchart is incomplete or non-understandable		
Design Pseudo-code	The flowchart and pseudo-code are correctly aligned, and they lead the robot to an accurate result	The pseudo-code is accurate, but it is not aligned to the flowchart design	The pseudo-code is not precise, and it is not aligned to the design	The pseudo-code is incomplete or non-understandable		
Implementation	The implemented program is accurate and is aligned to the design	The implemented program is accurate but not aligned to the design	The implemented program has some deviation of the chosen route and is not aligned to the design	The implemented program is not complete or it has syntax errors	N/A	

4.5 Validity and Reliability of the Instrument

A pilot was conducted with two students with a computer and information technology background. The pilot lasted 15 minutes for the pretest and 47 minutes for the posttest. Participants' impressions of the lab module were overall positive. Participants found some difficulties in attaining the exact pre-specified path. Participants found it enjoyable to work with the robot. These observations were used to refine the instructions and the learning materials.

5. Results

5.1 What are individuals' representational abilities for problem solving in the context of robotics challenges?

Table 2 depicts descriptive statistics for the individual rubric criterion as well as the total score. The results suggests a good performance by the students to move between different representations to solve a problem in robotics challenges. During the pretest, all participants (n = 44) were able to get a high average score (mean = 67.24%; SD = 15.81%) even though some of them (n = 28) did not have previous experience in programming courses. As mentioned earlier, the pretest assessment did not include the scores associated with the implementation task. The posttest score depicts even higher average scores both including the implementation score (mean = 82.39%; SD = 11.54%) and without the implementation score (mean = 78.60%; SD = 12.76%). The implementation score was 93.75%, with a moderate standard deviation of 12.21%. The results suggest that students with and without computing backgrounds were able to implement the robotics challenge.

Table 2. Pre and post –test performance to solve a robotics challenge problem

Test (N=44)		Mean	Mean (%)	SD	SD (%)
Pretest	Flowchart	2.64	65.91	0.97	24.17
	Analysis	2.64	65.91	0.75	18.75
	Design	2.80	69.89	0.73	18.35
	Total	2.69	67.23	0.63	15.81
Posttest	Flowchart	3.20	80.11	0.85	21.28
	Analysis	3.05	76.14	0.57	14.22
	Design	3.18	79.55	0.58	14.54
	Implementation	3.75	93.75	0.49	12.21
	Total w/o Implementation	3.14	78.60	0.51	12.76
	Total with Implementation	3.30	82.39	0.46	11.54

5.2 What is the effect of computational robotics challenges for improving individuals' computing representational fluency?

Figure 3 presents the comparison between the means of the pretest and posttest results. There are two different values related to posttest because it included an implementation question that was not part of the pretest. Therefore, both analysis with and without implementation scores are presented. Significant differences were found from pretest to posttest, both without implementation t(43)=-

5.7, p-value<0.001 and with implementation t(43)=-7.91, p-value<0.001. The test results suggest that the robotics activity increased students' computing representational fluency.

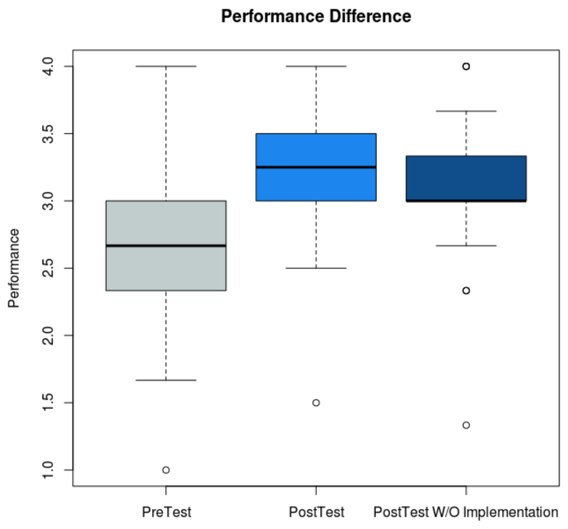


Fig.3 Comparison Pre and Post –Test performance to solve a robotics challenge problem

A correlational analysis was also performed to identify student representational fluency. Table 3 and Table 4 depict the correlations for the rubric criteria on the pretest and on the posttest correspondingly.

Table 3. Correlation among the rubric criteria on the pretest

	Flowchart	Design	Pseudo
Flowchart	1.00		
Design	0.26	1.00	
Pseudo	0.22	0.79	1.00

Table 4. Correlation among the rubric criteria on the posttest

	Flowchart	Design	Pseudo	Implement
Flowchart	1.00			
Design	0.32	1.00		
Pseudo	0.25	0.61	1.00	
Implement	0.35	0.46	0.49	1.00

The flowchart that was independent from the assignment moved from a weak-to-moderate correlation on the pretest to a moderate one on the posttest. The design, which consisted of a flowchart for the assignment, was strongly correlated to the pseudo-code written by the students both on the pretest and the posttest. Finally, the implementation showed a moderate-to-strong correlation to the design and to the pseudo-code criteria. The results suggest that students were able to build different representations for the phenomenon, both on the pretest and on the posttest.

5.3 Do individuals' background, academic level, and/or gender have an effect in their computing representational abilities for problem solving in the context of a robotics problem solving task?

Test results were also analyzed based on the independent variables Student Gender, Student Level, and Previous Experience in Programming Courses. Results suggest that there is no evidence of significant differences between genders F(43,1)=1.11, p-value=0.3, students' level F(43,1)=0.01, p-value=0.87, or previous experiences F(43,1)=0.15, p-value=0.7.

5.4 What are the individuals' perceptions about the usefulness of computational robotics challenges to learn algorithmic design and robotics?

Students' perceptions about usefulness related to the activity are described in Table 5. Engagement is highlighted as an important factor in this kind of activity (mean=84.09%; SD = 13.20). Also, although more than 60% of the participants did not have previous experience in programming courses (n=28), a large portion of the sample (74.09%) reported that the activity increased their interest in algorithms. Likewise, 78.18% of the participants felt that the activity increased their interest in robotics.

Table 5. Posttest students' perceptions related to the activity

Test	Mean	Norm Mean (%)	Std. Dev	Norm. Std. Dev (%)
Activities are engaging(N=44)	4.21	84.09	0.66	13.20
Activities increase interest in algorithms (N=44)	3.71	74.09	0.73	14.51
Activities increase interest in robotics (N=44)	3.91	78.18	0.73	14.66

6. Discussion and implications

From the analysis of student performance before and after being exposed to the learning experience, we can suggest that the design of learning activities guided by the use-modify-create pedagogy scaffolded the development of student computational representational abilities. This learning strategy might have supported learners in breaking down the activities in multiple steps so that they could make explicit connections between representations [35]. Since learning programming is a complex task[38], using multiple representations organized as Analysis, Design, and Implementation seemed to have helped students break down the problem in a step-by-step process. That is, by means of the scaffolding provided, students were able to decompose the posed problem into a flowchart to propose an initial solution [39]. Then, students transformed this representation into a pseudo-code and finally into a programming language. The scores for different representations, both on the pretest and on the posttest, showed a moderate-to-strong correlation, suggesting that high performer students in, for example, the flowchart design, also were high performers in the creation of the pseudo-code.

The artifacts the students produced and the progression they followed using one artifact and leveraging it to the creation of the next one is what we believe was particularly useful for them. Moving from natural language to flowchart, from flowchart to pseudo-code, from the actual code to testing, and the mappings between them, supported students in accomplishing their design task [30].

Findings also indicated no significant differences between pre- and posttest scores based on student academic level, gender, or disciplinary background. Based on these results, we speculate that the pedagogical strategy of use-modify-create coupled with robotics, can be used to integrate computational thinking concepts and skills with a diverse population of learners in terms of gender, interests, and expertise. On the other hand, since the interaction with multiple representations improves transfer [30], providing scaffolding for the students to go through these representations might also have had a positive impact.

In terms of motivation, several of the participants reported that the robotics-based activities were engaging. For instance, these students reported increased interest in both algorithm design and robotics. Furthermore, 60% of the students who had a non-computing background also reported positive perceptions of the usefulness of robotics challenges for their learning. These results are aligned with findings from other studies reporting that robotics activities are also useful for students with non-computing backgrounds [i.e., 19, 21]. Therefore, we speculate that the use of robotics can lower the barriers of entry into computing related fields.

6.1 Implications for Teaching and Learning

The implications for teaching and learning relate to the design of computational thinking learning experiences that are grounded in effective pedagogical methods and learning strategies. Firstly, this study provides a learning activity and learning assessments that can be easily adapted for learning purposes. Secondly, this study provides key insights into how literature from the learning sciences can be used to design learning experiences and their corresponding assessments. The emphasis on representational fluency, within the broader context of computational and algorithmic thinking, can guide the design of additional learning experiences following the process presented in this study.

This study also collected and analyzed evidence to weigh in on what kinds of learning resources we should bring to bear and the conceptual trade-offs they entail. The evaluation of learning materials suggests that, in a way, humans can build representational fluency effectively by exercising their physical intuitions. Specifically, robotics-based challenges can provide a tangible or sensory medium that, according to theories of embodied cognition, can foster development of conceptual understanding [40]. Therefore, we suggest that robotics can have a strong potential to serve as an effective and engaging vehicle to integrate principles and practices of computational thinking, such as algorithm design and principles of programming. Moreover, exposing students to an explicit representation and transformation processes scaffolded through the use-modify-create strategy can enhance their computational representational abilities.

6.2 Implications for Computing Educational Research

From a computing educational research perspective, this study portrays computational thinking as a practice of representation. Considering computational thinking in such way can allow researchers to investigate how students can manage complexity through a series of abstractions. Specifically, through the lens of representational fluency, the assessment of the learning process for this study was not only focused on the final product, but on the transitions from one representation to the next one. That is, the unit of analysis focused on (a) the process students followed in creating those artifacts and the mappings they produced between one representation to the other one (e.g. from a flowchart to a programming language) as well as (b) the outcome or final solution of the challenge presented to students (e.g. how the robot moved).

Computer science educators have called for the need to identify bridges between education research and computer science research with the goal to facilitate student learning of computing knowledge and practices [41]. This study provides a possible example of such process by integrating representational fluency to the design of a learning experience, and, then, to the investigation of its effectiveness.

The scholarship of teaching and learning implicates “engagement with research into teaching and learning, critical reflection of practice, and communication and dissemination about the practice of one’s subject” [42]. This study, in a way, went through a similar process by first designing the learning experience, then conducting the research and assessment components, disseminating the results, and then moving into iteration and revision to improve the learning materials and the research design. This process represents an initial stage toward a design-based research program that will investigate the role of representations in computing education. Design-based research approaches will allow us to understand learning in real-world practice [43]. It considers education as an applied field where researchers have transformative agendas [43]. As such, they develop contexts, frameworks, tools, and pedagogical models with the intent to produce new theories, artifacts, and practices that can impact teaching, learning, and engagement in naturalistic settings [43]. Therefore, design-based research will provide us with a series of approaches that allow us to “engineer” and at the same time study particular forms of learning that will be subject to test, revision, and iteration [44].

6.3 Limitations of the Study

Methodologically, this study had some limitations. One of the limitations in the research design was the lack of a control group. Another limitation included the sample size and the fact that participants were voluntarily recruited. This heterogeneous group led to small demographic subgroups that constrained the possible significant differences between them. Also, the study did not take place in a naturalistic classroom environment, where students are usually part of a longer learning process involving more variables. Therefore, the implementation of these practices should be further explored by means of more rigorous experimental designs to validate the learning experiences and the use of ethnographic methods to identify how students progress from one representation to another one; however, the results of this study empower us to implement, as future work, the robotics-based learning activities in classroom settings with a bigger and more homogenous sample of students and include a control group. It also provides us with a proof-of-concept that can allow us to explore computational thinking as a practice of representation.

7. Conclusion

This study proposed representational fluency as a research and learning framework that can allow the investigation of how people develop computational thinking. Under this perspective, this study presented the development of a learning module that integrated and validated pedagogical methods and scaffolding techniques to introduce computing principles and procedures by means of robotics-based challenges.

Findings from the implementation of these challenges suggest a positive impact on computational thinking in general and computational representational fluency specifically. Students with computing and non-computing backgrounds benefited from the use of robotics, and they performed equally in the posttest. These findings suggest that robotics can be used to learn computational thinking related concepts for designing, programming, and testing with a detailed level of abstraction. Results from this study also suggest that robotics may serve as a common theme to integrate STEM related concepts and computing and engineering skills. For instance, robotics can be used as viable source to teach students from both computing and non-computing backgrounds. Similarly, the robotics-based challenge can be adopted and adapted by educators for classroom use. It can also be used as a guide to develop new and more complex robotics-based challenges. The pedagogy presented here can also be used for other kinds of learning experiences not involving robotics.

The broader educational research community has made major calls to pursue discipline-based educational research [45], where we believe computer science education needs to be more strongly represented. The computer science community has also identified the need of more rigorous methodological approaches to pursue computer science education research [2, 3]. One of the key components toward a more rigorous path to discipline-based educational research in computer science is the consideration of theoretical foundations that can provide a perspective into how research has been grounded in literature and the scope and generalizability of the results [41]. Another key component would be the use of educational research findings to design computer science learning experiences [1]. A natural way to couple these two worlds could be by means of design-based research approaches that will allow educational practitioners and researchers to develop learning materials and pedagogical models with the intent of producing new theories, artifacts, and practices that can impact teaching, learning, and engagement in naturalistic settings [43].

8. References

[1] J. M. Clement, "A Call for Action (Research): Applying Science Education Research to Computer Science Instruction," *Computer Science Education,* vol. 14, pp. 343-364, 2004/12/01 2004.

[2] A. Pears and L. Malmi, "Values and objectives in computing education research," *ACM Transactions on Computing Education (TOCE),* vol. 9, p. 15, 2009.

[3] J. Randolph, *et al.*, "A methodological review of computer science education research," *Journal of Information Technology Education: Research,* vol. 7, pp. 135-162, 2008.

[4] J. M. Wing, "Computational thinking," *Communications of the ACM,* vol. 49, pp. 33-35, 2006.

[5] L. K. Soh, *et al.*, "Renaissance computing: an initiative for promoting student participation in computing," ed, 2009.

[6] [NRC], "Report of a workshop on the pedagogical aspects of computational thinking," National Research Council of the National Academies, Washington, D.C.2011.

[7] J. Cuny, *et al.*, "Demystifying Computational Thinking for Non-Computer Scientists," *Work in progress,* 2010.

[8] [CSTA]. (2012, March 15). *Operational definition of computational thinking*. Available: http://www.iste.org/Libraries/CT_Documents/Computational_Thinking_Operational_Definition_flyer.sflb.

[9] V. Barr and C. Stephenson, "Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?," *ACM Inroads,* vol. 2, pp. 48-54, 2011.

[10] [NRC], *Being fluent with Information Technology*: National Academy Press, 1999.

[11] J. Kramer, "Is abstraction the key to computing?," *Communications of the ACM,* vol. 50, pp. 36-42, 2007.

[12] R. Lesh, "Modeling students modeling abilities: The teaching and learning of complex systems in education," *Journal of the Learning Sciences,* vol. 15, pp. 45--52, 2006.

[13] M. Alhadeff-Jones, "Three Generations of Complexity Theories: Nuances and ambiguities," *Educational Philosophy and Theory,* vol. 40, pp. 66-81, 2008.

[14] E. Soloway and J. C. Spohrer, *Studying the novice programmer*: Lawrence Erlbaum Hillsdale, NJ, 1989.

[15] R. Lister, *et al.*, "A multi-national study of reading and tracing skills in novice programmers," *ACM SIGCSE Bulletin,* vol. 36, pp. 119-150, 2004.

[16] W. M. McCracken, *et al.*, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," *ACM SIGCSE Bulletin,* vol. 33, pp. 125-180, 2001.

[17] B. D. du Boulay, "Some difficulties of learning to program," in *Studying the novice programmer,* E. Soloway and J. C. Spohrer, Eds., ed: Lawrence Erlbaum, 1986, pp. 283-299.

[18] R. D. Pea and D. M. Kurland, "On the cognitive prerequisites of learning computer programming," 1983.

[19] D. C. Cliburn, "Experiences with the LEGO Mindstorms throughout the undergraduate computer science curriculum," in *Frontiers in Education Conference, 36th Annual,* San Diego, CA, 2006, pp. 1-6.

[20] F. Klassner and S. D. Anderson, "Lego MindStorms: Not just for K-12 anymore," *Robotics & Automation Magazine, IEEE,* vol. 10, pp. 12-18, 2003.

[21] B. S. Fagin, *et al.*, "Teaching computer science with robotics using Ada/Mindstorms 2.0," in *ACM SIGAda Ada Letters,* Bloomington, MN, 2001, pp. 73-78.

[22] D. Kumar and L. Meeden, "A robot laboratory for teaching artificial intelligence," *ACM SIGCSE Bulletin,* vol. 30, pp. 341-344, 1998.

[23] D. J. Barnes, "Teaching introductory Java through LEGO MINDSTORMS models," in *ACM SIGCSE Bulletin,* Cincinnati, Northern Kentucky, 2002, pp. 147-151.

[24] E. Bonakdarian and L. White, "Robocode throughout the curriculum," *Journal of Computing Sciences in Colleges,* vol. 19, pp. 311-313, 2004.

[25] J. O'Kelly and J. P. Gibson, "RoboCode & Problem-Based Learning: A non-prescriptive approach to teaching programming.," in *ITICSE '06 Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education,* Houston, TX, 2006.

[26] I. Angulo, *et al.*, "Competencias y Habilidades Con El Robot "Moway". ," in *VIII Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica - TAEE 2008,* Zaragoza, Espana, 2008.

[27] D. N. Perkins and G. Salomon, "Are cognitive skills context-bound?," *Educational researcher,* vol. 18, pp. 16-25, 1989.

[28] J. Bransford, *How people learn: Brain, mind, experience, and school*: National Academies Press, 2000.

[29] J. S. Brown, *et al.*, "Situated cognition and the culture of learning," *Educational researcher,* vol. 18, pp. 32-42, 1989.

[30] R. J. Spiro*, et al.*, "Cognitive flexibility, constructivism, and hypertext: Random access instruction for advanced knowledge acquisition in ill-structured domains," *Constructivism and the technology of instruction: A conversation,* pp. 57-75, 1992.

[31] I. E. Sigel, *Development of mental representation: Theories and applications*: Lawrence Erlbaum, 1999.

[32] Google. (2013, November 29). *What is CT? Exploring Computational Thinking*. Available: http://www.google.com/edu/computational-thinking/what-is-ct.html

[33] G. Cernosek and E. Naiburg, "The value of modeling," IBM developerWorks 2004.

[34] J. Kramer, "Abstraction-is it teachable? 'the devil is in the detail'," in *Proceedings. 16th Conference on Software Engineering Education and Training, 2003.(CSEE&T 2003).* , 2003, pp. 32-32.

[35] I. Lee, *et al.*, "Computational thinking for youth in practice," *ACM Inroads,* vol. 2, pp. 32-37, 2011.

[36] J. Malyn-Smith and I. Lee, "Application of the Occupational Analysis of Computational Thinking-Enabled STEM Professionals as a Program Assessment Tool," *Journal of Computational Science Education,* vol. 3, pp. 2-10, 2012.

[37] Robotis. (2013, July 22). *Robotis Inc.* Available: http://www.robotis.com/xe/

[38] J. Rogalski and R. Samurçay, "Acquisition of programming knowledge and skills.," in *Psychology of programming* J. M. Hoc, *et al.*, Eds., ed London: Academic Press, 1990, pp. 157–174.

[39] S. P. Lajoie, "Extending the scaffolding metaphor," *Instructional Science,* vol. 33, pp. 541-557, 2005.

[40] Z. C. Zacharia, *et al.*, "Is physicality an important aspect of learning through science experimentation among kindergarten students?," *Early Childhood Research Quarterly,* vol. 27, pp. 447-457, 2012.

[41] M. Daniels and A. Pears, "Models and Method for Computing Education Research," in *Proceedings of the 14th Australasian Computing Education Conference (ACE2012),* Melbourne, Australia, 2012.

[42] M. Healey, "Developing the scholarship of teaching in higher education: a discipline-based approach," *Higher Education Research and Development,* vol. 19, pp. 169-189, 2000.

[43] S. Barab and K. Squire, "Introduction: Design-Based Research: Putting a Stake in the Ground," *The Journal of the learning sciences,* vol. 13, pp. 1-14, 2004.

[44] P. Cobb, *et al.*, "Design experiments in educational research," *Educational researcher,* vol. 32, pp. 9-13, 2003.

[45] [NRC], Discipline-Based Education Research. Understanding and Improving Learning in Undergraduate Science and Engineering. Washington, D.C.: National Academies Press, 2012.

[46] A. Rubin, “Statistics for Evidence-Based Practice and Evaluation”; Cengage Learning: Belmont, CA. 2009.