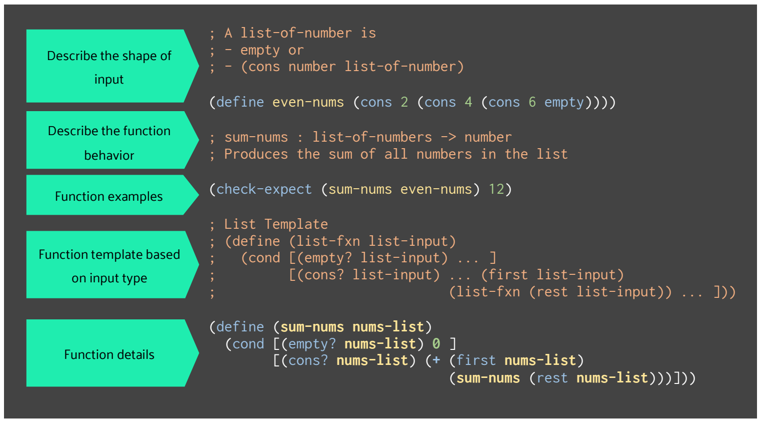
Computing now permeates many aspects of people's everyday lives. Social media platforms break distance barriers in communication, online systems make both regular and critical engagements more accessible, and even media-rich online learning has transformed the access and delivery of education. Because of these, more people want to learn computing – some to get into computing-centric fields for career advancement, others to advance fields that improve our quality of life. CS education is thus becoming an integral part of most educational systems to help meet the demand of a growing population of computing learners, but as a young field, little is still known about what makes an "effective" CS education. We see this in prevalent CSEd problems such as dismal student performance in assessments and curricula that focus on the constructs and features of tools (e.g. programming languages) but lack explicit instruction of techniques that leverage these features to solve problems, among others. Researchers in CSEd still have a lot of work to do towards understanding the factors revolving around CS education: cognitive, pedagogical, socioeconomic, and sociocultural underpinnings, and many others [9]. My work builds on this understanding through the development of learning frameworks that capture learners' program-design skills and how learners use these skills in their programming practice.

My research agenda primarily stems from my own direct experiences with problems in computing education, both in CS-centric contexts and in my prior work in computing in non-CS-centric fields. My research agenda is composed of four main themes around (1) **building a CS-centric learning theory** to form an understanding about how students learn to design programs, (2) using insights from this learning theory to identify ways to **help CS educators teach effectively**, (3) developing CS learning theories further to identify how early-CS education can **support learning beyond CS1**, and (4) using what we know about CS education to **bring computing education to non-CS majors and learners in marginalized communities**. I describe here my current work and outline plans for future research directions, as shaped by my PhD work and prior research experience in various computing-application fields.

1: My PhD Research – Building a Program-Design Learning Theory in Early-CS

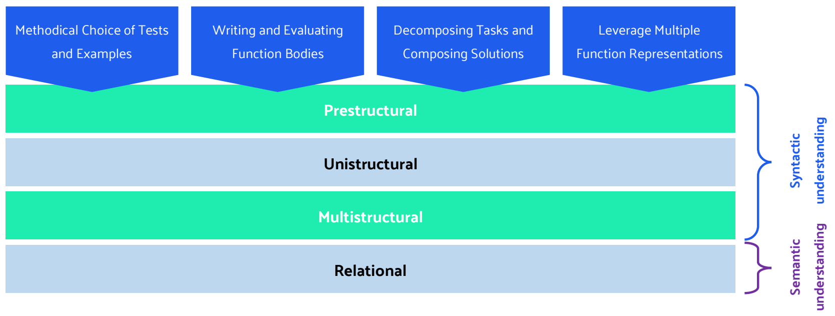
In my dissertation work, I use methods from human-factors research, educational psychology, and cognitive psychology to directly study early-CS students' processes when designing programs. Students in my studies are taught to design programs through the *How to Design Programs* (HtDP) curriculum [7], which uses a unique pedagogy that systematizes the design process. The HtDP process scaffolds students' program-design practice by gradually and increasingly illuminating information about the problem-domain at hand, leading up to the development of a final program artifact. At the heart of this process is the use of a design-pattern that reflects the structure of input data in the problem-domain; this process challenges traditional practices in introductory-level CS instruction that mainly focus on the mapping of language-centric implementations to problems [6]. A specific research question I thus focus on in this work is: how do students use the HtDP process to design programs?


Describe the shape of input
Describe the function behavior
Function examples
Function template based on input type
Function details

```
; A list-of-number is
; - empty or
; - (cons number list-of-number)

(define even-nums (cons 2 (cons 4 (cons 6 empty))))

; sum-nums : list-of-numbers -> number
; Produces the sum of all numbers in the list

(check-expect (sum-nums even-nums) 12)

; List Template
; (define (list-fxn list-input)
;   (cond [(empty? list-input) ... ]
;         [(cons? list-input) ... (first list-input)
;                                (list-fxn (rest list-input)) ... ]))

(define (sum-nums nums-list)
  (cond [(empty? nums-list) 0 ]
        [(cons? nums-list) (+ (first nums-list)
                              (sum-nums (rest nums-list)))]))
```

The HtDP pedagogy systematizes the program-design process through a sequence of concrete steps.

My work to date has resulted in the development of a novel multidimensional framework that (1) captures the **skills** that students use in designing programs, and (2) the various levels of students' **performance** in each skill [1]. My paper detailing the development of this framework was awarded **Best Paper** at the 2017 Koli Calling International Conference on Computing Education Research. I have also obtained preliminary data around **how students use these skills in practice**, how their skill-use influences **how they structure program artifacts**, and **how their performance in these skills evolve** across a CS1-level course [1, 2, 8]. There is still very limited understanding about the acquisition, use, and development of programming knowledge, and there is not enough learning theory in CS education research to explain or understand how our current modes of instruction support CS learning [9]. The results of my work, developed from direct observations of students learning within a concrete program-design curriculum, contribute to this development of CS-centric learning theories, and can be used to inform the design of instructional material, curricula, and assessment.


Methodical Choice of Tests and Examples
Writing and Evaluating Function Bodies
Decomposing Tasks and Composing Solutions
Leverage Multiple Function Representations
Prestructural
Unistructural
Multistructural
Relational
Syntactic understanding
Semantic understanding

An overview of the program-design skills framework that captures students' skills and variations in performance within each skill. See Castro and Fisler, 2017 [1] for the full detailed framework.

2: Evidence-based Design of Early-CS Instruction Material

The program-design skills framework I have developed enables a more nuanced assessment of students' growth in their learning of program-design in early-CS courses. A research direction I envision around this is exploring how the framework enables educators to assess early-CS instructional material to concretely identify content that support the development of particular design skills. One way of doing this is through the development of collections of **programming problems that are aligned towards specific design-skills** and **appropriately-scaled towards different levels of understanding**. This touches on an important concern in CSEd research: there is little understanding about the cognitive complexity of CS examples and problems used in CS classes [9, 10, 11], and this lack of understanding about the problems we use in teaching may affect how well we assess students' learning. By gaining an understanding of the inherent expectations and knowledge dependencies that problems place on students, we can better design assessments that target concrete skills and knowledge we want students to learn, as well as develop materials for learners of varying experience or backgrounds. These projects open opportunities for fruitful collaboration between CS education, learning science, and education researchers in other fields.

3: Program-design Skill Transfer and Acquisition Beyond Early-CS and Across Learning Contexts

One of the primary goals of CS1 is to provide a foundation that supports student learning beyond CS1. This means that early-CS learning theories should help us gain an understanding of how the skills and knowledge gained by students in early-CS **help them assimilate more advanced concepts**. Much of my PhD work has focused on learners in HtDP-based CS1 contexts. I would like to explore how the skills and programming process patterns I have observed transfer to, or differ from other CS-learning contexts. These contexts could be advanced CS courses (e.g. CS2), courses using different programming languages and paradigms, or learning environments that use different instructional strategies such as flipped classrooms and peer-learning.

These open up interesting questions around (1) how and in what ways program-design skills acquired in early-CS support the learning of advanced programming concepts, (2) the interactions of language and paradigm choices with skill acquisition – what skills, variations in skill performance, or programming processes these may introduce, and (3) whether different instructional strategies encourage different ways of learning program-design. Findings we gain from addressing these questions will help build CS-centric learning theories further, especially around the affordances of various CS-learning contexts. The development of these learning theories will also **provide strong theoretical foundations that can inform or support the design of tools for CS-learning**, such as IDEs, tutoring systems, and peer-learning systems to name some, and understand better the ways in which these tools support learning. These projects pull in field knowledge from CSEd, HCI, and the learning sciences and provides a collaborative venue for students and experts in these fields.

4: Bringing Computing Education Beyond CS-centric Contexts

4.1: Computing Education for Non-majors

As an instructor at the Ateneo de Manila University and University of the Philippines Los Baños, I taught CS courses attended by both CS and non-CS majors, including an interdisciplinary computing course in health informatics. As a teaching assistant at WPI, this diversity of student majors were also present in the CS1/2 courses I worked in, with majors varying widely across CS, robotics engineering, mathematics, bioinformatics, and many others. This variety inevitably meant that I worked with students with a diversity of cultural and social capital for learning the material. While my current work has focused on studying the nuances of how university-level students specifically learn and use program-design skills, this is encompassed in a larger idea of exploring the nuances of how people, potentially from diverse contexts, learn computing.

In developing the program-design skills framework [1] for my PhD thesis, I also noted themes from my data that point to external factors such as value judgments and quality attributes of artifacts that appear to influence students' design decisions around programming. This offers preliminary insights that point to different students' potential priorities, motivations, expectations, or values that they pull in when they participate in computing-centric learning environments. These factors affect how people are learning computing, their motivations for participating in learning, and how they align their identities with the learning goals of the environments they are immersed in [5].

This opens several questions: how do we design learning environments, particularly computing or interdisciplinary courses especially for non-CS majors, who may have different expectations and motivations of computing from those of CS-majors? **How do we design or shape a learning environment, taking into consideration various factors such as students' values, priorities, or backgrounds?** How might computing learning theories around such learning environments look like and what does this tell us about how to teach computing within these environments? A health informatics course, for example, tackles issues around system and interaction design, public health, and the design of protocols to train end-users to use health systems. We can explore how students navigate around these needs and priorities, how their values, prior knowledge, and experiences shape their learning, and how we might use findings from these to **design computing courses that better cater to the needs of non-CS majors**.

4.2: Computing Education for Marginalized Communities

A cause I am personally strongly passionate about is bringing computing education to marginalized and low-income communities. This is fueled by my experiences and interactions with communities outside of academic institutions. In the Philippines, I volunteered in centers for out-of-school youth to teach English and the use of computing applications. In my external projects as an instructor, I did front-facing user-research on health-technology use with people and healthcare practitioners in remote villages who have had limited exposure to computing and computing education. In a similar way as students who are non-CS majors, learners from low-income communities bring with them intrinsic qualities that may affect the dynamics of learning in a computing education environment. Low-income learners have been found to face challenges in classrooms because of their lack of access to technology, such as computers, in their own homes. This in turn affects educators' use of computing in classrooms to augment learning, as their students are not as immersed in the affordances offered by computing tools [4]. How then do you extend the reach of computing education to low-income communities and how do you help these communities integrate computing education into their existing learning context?

One of the ways I've envisioned working on this problem is by integrating "offline" computing learning activities (such as ones from CS Unplugged [3]), with low-cost and simple interfaces (e.g. low-cost robots or screen-based interfaces). This meets the students halfway through an offline learning activity to teach computing concepts (e.g. control flow), which can be applied in a tangible interface tool to show concepts "in action" (e.g. input a sequence of directional instructions for an object to follow), thus **providing a low-cost and immersive computing-learning experience**.

This research direction builds on existing ideas and/or tools, but what makes this important in my opinion, is the focus of **making computing education available to a sector of society that does not have an easy access to computing**. Relative to my current work, it would be interesting to see the kinds of computing skills learners gain from the use of these tools and how their lived experiences might influence how these skills are learned. We can also explore the ways in which the integration of instruction and tools in this project supports the learning of computing concepts beyond the affordances offered by such efforts.

Summary

Understanding (1) the **nuances of how learners learn computing** and (2) how we can use this knowledge to **design better instruction and learning environments** are recurring themes in my research agenda. My work drives the research community forward through my contributions to the fields of computing education and the learning sciences, and by integrating my work with other fields such as human-computer interaction, health informatics, and education. I am passionate about working towards a deeper understanding of the human factors around learning and computing. Such an understanding will help us develop more meaningful ways of teaching computing so that learners benefit most from the learning experience.

References

[1] **Francisco Enrique Vicente Castro** and Kathi Fisler. 2017. Designing a Multi-faceted SOLO Taxonomy to Track Program Design Skills Through an Entire Course. In Proceedings of the 17th Koli Calling Conference on Computing Education Research (Koli Calling '17), 10–19.

[2] **Francisco Enrique Vicente Castro** and Kathi Fisler. 2016. On the Interplay Between Bottom-Up and Datatype-Driven Program Design. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16), 205–210.

[3] CS Unplugged: Computer Science Without a Computer. http://csunplugged.org/

[4] Linda Darling-Hammond, Molly B. Zielezinski, and Shelley Goldman. 2014. Report: Using Technology to Support At-Risk Students' Learning. Stanford Center for Opportunity Policy in Education.

[5] Kayla DesPortes, Monet Spells, and Betsy DiSalvo. 2016. The MoveLab: Developing Congruence Between Students' Self-Concepts and Computing. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16), 267–272.

[6] Michael de Raadt, Mark Toleman, and Richard Watson. 2007. Incorporating Programming Strategies Explicitly into Curricula. In Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88 (Koli Calling '07), 41–52.

[7] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2001. How to Design Programs: An Introduction to Programming and Computing. MIT Press. http://www.htdp.org/

[8] Kathi Fisler and **Francisco Enrique Vicente Castro**. 2017. Sometimes, Rainfall Accumulates: Talk-Alouds with Novice Functional Programmers. In Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17), 12–20.

[9] Mark Guzdial. Communications of the ACM. Learning Computer Science is Different than Learning Other STEM Disciplines. https://cacm.acm.org/blogs/blog-cacm/224105-learning-computer-science-is-different-than-learning-other-stem-disciplines/fulltext

[10] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. In Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '01), 125–180.

[11] Ian Utting, Allison Elliott Tew, Mike McCracken, Lynda Thomas, Dennis Bouvier, Roger Frye, James Paterson, Michael Caspersen, Yifat Ben-David Kolikant, Juha Sorva, and Tadeusz Wilusz. 2013. A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations. In Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports (ITiCSE -WGR '13), 15–32.