

Towards a Theory of Program-Design Learning Through HtDP

Francisco Castro

Advisor: Kathi Fisler



How are students doing in intro-program-design?

Problems in intro-CS are well-documented: high attrition, inadequate programming proficiency, dismal performance in assessments

- Affective factors: pervading “CS is hard” notion, lack of interest/motivation
- Socio-cultural, socio-economic factors
- Language-centric curricula, absence of explicit instruction of program-design techniques and strategies
- Students unable to decompose problems into subproblems and recombine pieces into solutions

functional programming, because of the data structures she's using, or because she's teaching higher-level functions? We don't really know what makes programming so hard, and we don't yet have enough theory to explain why it works when we get it right.

- Mark Guzdial (BLOG@CACM: Learning Computer Science is Different than Learning Other STEM Disciplines, Jan. 5, 2018)

Classic example of CSEd problem that students struggle to solve: The Rainfall Problem

The Rainfall problem

Design a program called `rainfall` that consumes a list of numbers representing daily rainfall readings. The list may contain the number **-999 indicating the end of the data of interest**. Produce the **average** of the non-negative values in the list up to the first -999 (if it shows up). **There may be negative numbers other than -999** in the list representing faulty readings. If you **cannot compute the average** for whatever reason, return -1.

Research has shown Rainfall is hard for intro-level students

Some ongoing research: how to incorporate design strategies/patterns in early-CS curricula so students come out of early-CS courses able to solve such multi-task problems

Our flavor for program-design instruction: the HtDP project

(Multiple) Tasks (!)

- Truncate at -999
- Skip negative values
- Average non-negative values
 - Sum non-negatives
 - Count non-negatives
 - Divide sum by count
- Guard from zero division

How to Design Programs (HtDP)

- Systematizes program-design through a step-by-step *design recipe* (more later)
- Used in a number of universities/colleges, high schools^[1, 2], K-12 programs^[3]
- Reported success at the college level --- students show good programming habits and more likely to receive higher grades (than students in other curriculum)^[2]

However: No attention to *cognitive* underpinnings of how HtDP works;
limited formal evaluation of how students work with HtDP

General question: How do students use the HtDP process to design programs?



We looked at existing models of program-design

Rist's model: worked on the idea of *schema* (a mental organization of knowledge) *retrieval*

Rist developed this model from observing novices program imperatively in Pascal

- **Retrievable schema:** top-down, straightforward development of solution after retrieving from memory (“retrieval state”)
- **No retrievable schema:** bottom-up development from an identifiable computation (“creation state”)

Retrieval



```
sum = 0
for each num in input_list:
    if num >= 0:
        sum = sum + num
return sum
```

Creation



We weren't sure how HtDP fit this model...

What is **How to Design Programs** (HtDP)?

Design recipe

Describe the shape of input

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

Example:
Write a function
to sum a list of
numbers

Describe the
function behavior

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

Function examples

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

Function template
based on input
type

```
(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)) ... ]))
```

Function details

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

*Note: Semicolon (;) used for comments

Rist model and HtDP??

We weren't sure that HtDP fit Rist's model...

1. Don't start with a program --- start with steps that illuminate more information about the problem
2. Don't start with a computation --- design recipe gives you a pattern to start with, the template, based on the input type in the problem

Retrieval



```
sum = 0
for each num in input_list:
    if num >= 0:
        sum = sum + num
return sum
```

Creation



What is the interaction between this programming model and the HtDP process?

Exploring interplay of HtDP with Rist's model: Adding-machine study


What did we do?

- We gave HtDP-trained CS1 students the Adding Machine programming problem and video-recorded the IDE window as they worked on the problem
- We open-coded a sample of the submissions to identify how students were using HtDP to solve the problem

Design a program called adding-machine that consumes a list of numbers and produces a list of the sums of each non-empty sublist separated by zeros. Ignore input elements that occur after the first occurrence of two consecutive zeros.

Input: (list 1 2 0 7 0 5 4 1 0 0 6)

Output: (list 3 7 10)

sum ()

** familiar data-type, but needs more advanced concepts: e.g. parameter that accumulate data, reshape the data, etc.

What did we find out?

1. After writing a template for the Adding Machine function, students wrote expressions that took on specific tasks within the template

- Tasks students wrote expressions for: sum, single-zero, double-zero
- However, unclear when students were in a “creation” state ---
e.g. summing is standard HtDP problem, terminating at single- and double-zero patterns resembled base case tasks (which usually treated only the empty list)

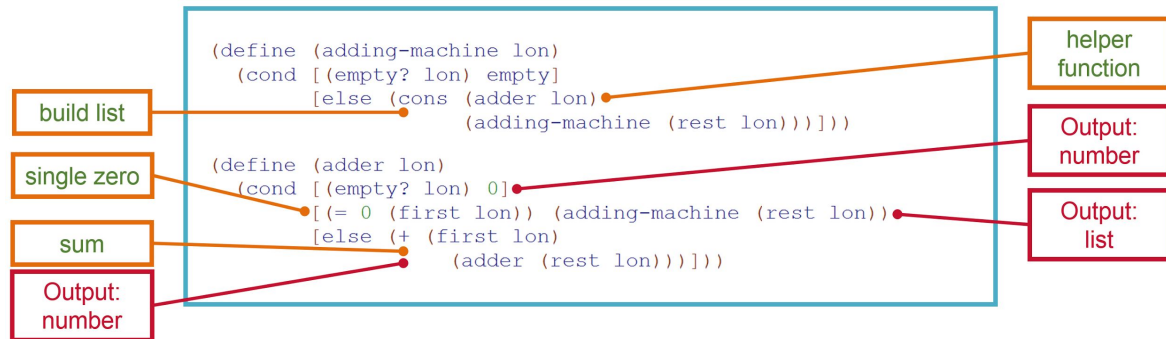
2. Some students did not decompose the problem and erroneously combined multiple tasks in a single template instance

e.g. sum and double-zero tasks in one function:
return zero to terminate sum, return a list to terminate double-zero (output inconsistency)

- Students seem to not have learned when to decompose problems into multiple template instances...
- ... Or may lack understanding that one template only returned one output type

What did we find out? (2)

3. Some students who tried to decompose the problem using helpers failed to compose solutions due to a verbatim use of the template



- Call helper that summed values of a sublist; BUT, recursive call still took the rest of the entire input list instead of just the suffix without the sublist
- Seemed to be a verbatim use of the template without a clear understanding of what parts of the data still need to be processed in the recursive calls

Next step?

Adding-machine study data was from a single point of the course --- observations at that particular point of learning

We wanted to understand further how students' use of HtDP *evolved* across a CS1 course

General question:

Do we see an evolution of how students are using HtDP to solve programming problems?



Exploring evolution of students' HtDP use: Multi-session study

What did we do?

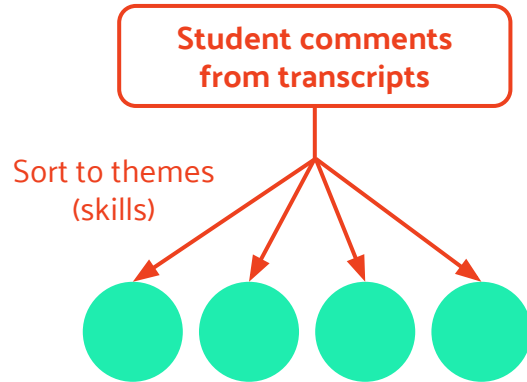
- We interviewed students about how they used HtDP to solve homework problems (3 sessions)
- In the last session, we did a think-aloud + post-interview session. Students solved the Rainfall problem while articulating their thought-process
- We open-coded a subset of the transcripts to figure out how students used HtDP to solve programming problems

Session 1	Activity	Interview* on homework problem
	Topic	List of tuples/structures (sum cost of ads for a political candidate)
Session 2	Activity	Interview* on homework problem
	Topic	n-ary trees (check oxygen levels in a river system)
Session 3	Activity	Think-aloud and post-interview*
	Topic	Rainfall (average non-negative numbers from a list until sentinel)

*Interview questions asked students to describe how they approached problems and their use of the design recipe






What did we find out?

1. We identified *concrete skills* that students demonstrated while using HtDP

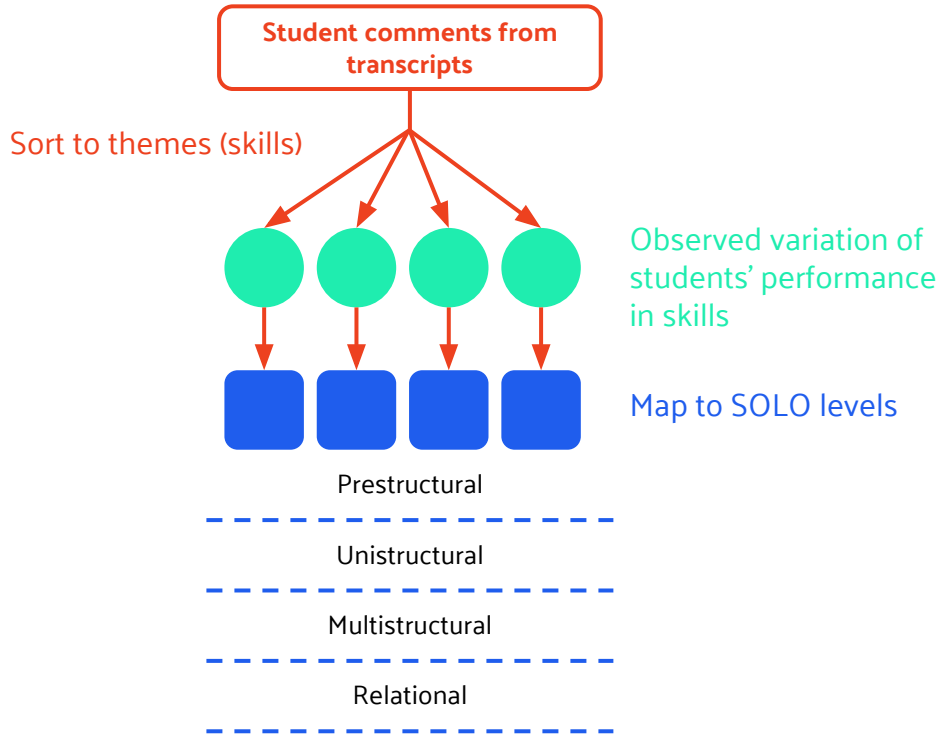


2. We also saw that students' performance in these skills varied; the variation resembled *SOLO-like* levels

Structure of Observed Learning Outcomes (**SOLO**) captures different levels of complexity:

	Pre-structural	No understanding
	Uni-structural	Understand a single aspect
	Multi-structural	Understand several aspects independently
	Relational	Inter-operation of several aspects
	Extended Abstract	Generalize to a new domain

3. After mapping students' comments to skills we identified, we mapped the comments to SOLO levels, resulting in a *multi-dimension framework* that captures (1) the skills observed and (2) the variations of students' performance for each skill



SOLO level	Methodical choice of tests and examples
Prestructural	Does not know how to write tests; misses the input/output structure of tests
Unistructural	Able to write tests; descriptions of tests do not explain the purpose of the test(s); does not express the idea of varying test scenarios
Multistructural	Able to write multiple tests; articulates the purpose of individual tests but does not articulate any relationship between or collective purpose for the tests
Relational	Able to write tests; identifies a collective purpose for the tests, i.e. boundaries, edge cases, test space coverage, but limited within the context of the problem

The resulting framework

SOLO level	Methodical choice of tests and examples	Writing and evaluating function bodies	Decomposing tasks and composing solutions	Leverage multiple representations of functions
Prestructural	Does not know how to write tests; misses the input/output structure of tests	Does not know how to define a function	Does not identify relevant tasks for a problem	Just dives in and writes code; uses only a single representation
Unistructural	Able to write tests; descriptions of tests do not explain the purpose of the test(s); does not express the idea of varying test scenarios	Able to define functions in a simple context - uses primitive operations on primitive types in a function body	Able to identify relevant tasks but no reflections of separate tasks when talking about the code	Blindly follows the design recipe; sees each function representation as independent of others
Multistructural	Able to write multiple tests; articulates the purpose of individual tests but does not articulate any relationship between or collective purpose for the tests	Able to define functions whose bodies contain nested non-primitive expressions or function calls, but does not articulate the semantics of how the results of calling a function return to the calling context	Able to identify relevant tasks; articulates the delegation of tasks into separate functions but fails to articulate how to effectively compose the tasks in a way that solves the problem	Articulates a sense of the function representations talking about or referring to the same computation
Relational	Able to write tests; identifies a collective purpose for the tests, i.e. boundaries, edge cases, test space coverage, but limited within the context of the problem	Able to define functions whose bodies contain nested non-primitive expressions or function calls and is able to articulate the semantics of how the results of calling a function return to the calling context	Able to identify relevant tasks; articulates the delegation of tasks into separate functions and can articulate how to effectively compose the tasks in a way that solves the problem	Articulates a mechanism through which function representations are related, e.g. template uses types to drive the code structure, execution of a program connects to a test space, etc.

Syntactic

Semantic

The relational level establishes logical connections between schema/artifacts from prior levels

Initial validation of the framework

We applied the framework to all transcripts (3 sessions x 13 students) and categorized student data even beyond the sample

student5

S #	Tests	Fxns	Decmp	FxnRep
1	R	M	U	U
2	R	R	R	U
3	R	R	R	M

- Students show performing at different levels for different skills at a given time
- ✓ Multidimensional taxonomy captures variances in students performance in different skills

student1

S #	Tests	Fxns	Decmp	FxnRep
1	U	M	U	U
2	R	R	M	U
3	U	M	M	U

- Some students show non-monotonic progression of skills through the sessions
 - Skills may not have been internalized well
 - Problems may push students towards particular levels
 - Drops may reflect the problem complexity at which students can apply skills

Exploring students' use of HtDP: Rainfall think-aloud study

Like the *Adding-machine study*, the think-aloud data we collected allowed us to reconstruct how students were using HtDP to solve a programming problem, and also captured students' thought processes

What did we do?

- We developed narratives of how students solved the Rainfall problem from the think-aloud and post-interview data



What did we find out?

1. Students who simply copied the template struggled more with changing their solution structure later on than those who wrote the template “as they went”

2. Students who only had a syntactic-level understanding of the accumulator pattern (using parameters to accumulate a running value), struggled to adapt it to the needs of the problem

StudA: “I guess [the hardest part] was trying to figure out how to work in the -1 with the accumulator there because I didn’t know where to put it [...] all the examples we put the accumulator after empty [...] but in this one the answer wasn’t stored in the accumulator.”

3. Students who made connections between parts of their code (e.g. parameters, functions) to specific tasks they identified were able to produce more correct code

Common observation from this study and the *Adding-machine study*:

Students who exhibit a mostly syntactic-level understanding of using HtDP templates (and its variations) struggled to solve multi-task problems.

2 primary outputs from completed studies:

1. SOLO-based framework that formalizes skills around students' use of HtDP and the variations of student performance of each skill
2. Narrative accounts and descriptions of students' use of HtDP to solve programming problems, with a particular focus on template use

Back to the original question:

How do students use HtDP to design programs?

Based on these studies, we have distilled this bigger question into two research questions that we have partially addressed through the studies so far ---



RQ1. How do students' skills around the use of HtDP design steps evolve through a course?

1.1. What skills do learners exhibit in their use of HtDP?

1.2. How do students' performance in the skills evolve through CS1?

1.3. In what ways do the skills interact with each other? Which skills, if any, seem to develop earlier than others?

RQ2. How do students adapt their use of HtDP templates to problems with multiple task components?

2.1. How do students perceive templates and their role in program design? Do they see templates as a *traversal schema*, a container schema, or something else?

2.2. When dealing with a multi-task problem, do they recognize when a naive use of the template will not suffice? How do they address this --- do they *allocate* tasks to multiple template instances, *mutate* the template, force the *naive use* of the template, or something else?

Goal: Understand how students use the HtDP process to design programs

RQ1.
Identify skills around HtDP use and evolution of students' performance

	Test	Fxns	Dcmp	Fxn Reprs
Pre				
Uni				
Multi				
Rel				



Does the framework capture similar nuances as HtDP experts when used to assess skill levels?

```
(define (adding-machine lon)
  (cond [(empty? lon) empty]
        [else (cons (adder lon)
                     (adding-machine (rest lon)))]))

(define (adder lon)
  (cond [(empty? lon) 0]
        [(= 0 (first lon)) (first (rest lon))]
        [else (+ (first lon) (adder (rest lon)))]))

(define (adding-machine lon)
  (cond [(empty? lon) empty]
        [(and (= (first lon) 0) (= (first (rest lon)) 0)) empty]
        [(= (first lon) 0) empty]
        [else (cons (cond
                     [(empty? lon) 0]
                     [else (cond
                              [(= (first lon) 0) 0]
                              [else (+ (first lon) (adding-machine (rest lon)))]))]
                    (adding-machine (rest lon)))]))
```

RQ2.
Describe accounts of HtDP use (template)



Can the skills framework be used as a predictor of how students might use HtDP to design programs?

Do we see the same skill-profiles (later)/HtDP-use patterns among students from other HtDP institutions/on other multi-task problems?

Proposed study: Expert validation of SOLO skills framework

Goal: Check whether our skills framework captures similar nuances to those raised by experienced HtDP instructors

What do we plan to do?

- Give HtDP instructors: [1] student data (transcripts + code) and [2] likert-scale survey to assess students' design skills based on student data
- Check:
 - (1) How similar are the SOLO-based ratings with instructors' ratings of skills?
 - (2) Do the descriptions of skill levels in the framework capture nuances in instructors' rating explanations?

Rating scale:

- 0: None
- 1: Little to none
- 2: Fragmented, but present
- 3: Strong
- 4: Applies beyond current problem

Student: _____	Skills			
Session Number	Methodical choice of tests and examples	Writing and evaluating function bodies	Decomposing tasks and composing solutions	Leveraging multiple representations of functions
Explain your rating				

This will help us determine how well our SOLO skills framework can be used as an instrument for assessing students' program-design skill levels

Proposed study: Explore interactions between SOLO skill profiles and HtDP-use patterns

Goal: Determine utility of the SOLO skills framework as a predictor of how students might use HtDP to design programs

What do we plan to do?

- Conduct think-aloud/interview-style sessions with new student cohort and problem
- Use the SOLO skills framework to assess students' skill levels; develop narratives to describe accounts of HtDP use

Can the skills framework be used as a predictor of how students might use HtDP to design programs?

studentX

Tests	Fxns	Decmp	FxnRep
R	R	M	U

*Skill profile: “snapshot” of students' SOLO levels across different skills

Do students with similar skill profiles exhibit similar errors? Which kinds of errors (e.g. programming, logic errors, HtDP use errors, etc)?

Do students with similar skill profiles structure solutions similarly? Use templates similarly?

Proposed stud(y/ies): Replicate studies on HtDP cohorts from other institutions

Goal: Test our analyses in other HtDP-trained cohorts to further validate findings

What do we plan to do?

- Replicate data collection protocol from the *Multi-session* study
- Use the SOLO skills framework to assess students' skills; develop narratives to describe accounts of HtDP use

Piecing everything together: Understand how students use the HtDP process to design programs

RQ1: How do skills evolve?

- Identify the skills
- Figure out the performance levels within the skills
- ** Make sure they aren't problem-dependent [TODO]
- ** Make sure the skills and levels are meaningful from an instructor's perspective [TODO]
- ** Explore how the different skills co-evolve during a course [TODO]

Outcome: An understanding of skills that HtDP fosters, a sense of how to measure them, and a sense of the extent to which they are problem dependent

RQ2: How to adapt use of HtDP to multi-task problems?

- ** Observe use of skills on various multi-task problems [TODO]
- ** Explore whether skills differ across different kinds of multi-task problems or student preparation to solve them [TODO]

Outcome: Data that HtDP instructors can use to select pedagogy and assessment towards different kinds of multi-task problems

Overall outcome: A better understanding of the skills that students develop and apply after studying through HtDP.