Towards Understanding Human Mistakes of Programming by Example: An Online User Study

Tak Yeon Lee
Human-Computer Interaction Lab
Computer Science Dept.
University of Maryland
College Park, MD, USA
reflect9@gmail.com

Casey Dugan
IBM Research
Cambridge, Massachusetts, USA
cadugan@us.ubm.com

Benjamin B. Bederson
Human-Computer Interaction Lab
Computer Science Dept.
University of Maryland
College Park, MD, USA
bederson@umd.edu

ABSTRACT

Programming-by-Example (PBE) enables users to create programs without writing a line of code. However, there is little research on people's ability to accomplish complex tasks by providing examples, which is the key to successful PBE solutions. This paper presents an online user study, which reports observations on how well people decompose complex tasks, and disambiguate sub-tasks. Our findings suggest that disambiguation and decomposition are difficult for inexperienced users. We identify seven types of mistakes made, and suggest new opportunities for actionable feedback based on unsuccessful examples, with design implications for future PBE systems.

Author Keywords
Programming-by-Example; User study

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous

INTRODUCTION

The goal of programming by example (PBE) is to enable ordinary people to automate complex and repetitive tasks, and it has even made its way into commercial products such as Microsoft Excel's FlashFill [2]. However, guiding inexperienced users on how to provide high-quality examples is still an open-ended research question. To create high-quality examples, users need to consider two requirements: (1) **disambiguation**, and (2) **decomposition**. First, users must be able to provide diverse cases to disambiguate the operation they want to create from other operations the PBE engine could infer. Second, to create operations for complex tasks, users need to decompose those tasks into small sub-tasks that the PBE engine can (more easily) infer. Both disambiguation and problem decomposition are challenging computational thinking skills and are often part of required training for computer science and engineering students.

For this paper, we conducted an online user study with participants recruited from Amazon Mechanical Turk (AMT) who were asked to complete 6 tutorials and 5 main tasks using our PBE system. Our research focuses on examining the behavior of ordinary people providing input and output examples, managing steps and cases for decomposition and disambiguation, and making and fixing mistakes. To provide recommendations for PBE tool designers, we also designed two feedback mechanisms, and compared their impact on the main task success rate. A total of 161 users participated in the study, and 30 of them successfully completed all five main tasks.

Our findings suggest that disambiguation and decomposition are difficult for even highly-motivated AMT workers, and for those that had practiced all required subtasks during the tutorials. We report seven types of mistakes identified from unsuccessful trials. We also determined that that those unsuccessful trials contain meaningful information about users' intent and misunderstandings about PBE. Under the actionable feedback condition, participants received context-aware suggestions based on the information from unsuccessful trials, and outperformed other participants.

IUI 2017, March 13-16, 2017, Limassol, Cyprus
© 2017 ACM. ISBN 978-1-4503-4348-0/17/03$15.00
DOI: http://dx.doi.org/10.1145/3025171.3025203

RELATED WORK

There exists a large body of research on PBE systems, as summarized in [1,8]. User studies on PBE systems reveal usability issues and challenges, summarized by Lau [7] and Gulwani [4]. A typical PBE system generates multiple operations that are consistent with the examples provided by the user. To resolve the ambiguity, researchers have proposed a few interaction models. For example, Wrangler [6] lets users choose an operation among top candidates, FlashProg [9] also makes suggestions to users to provide specific examples for resolving ambiguity. A few PBE systems [6,11] support decomposition by allowing users to create multiple operations one-by-one. However, users of such systems are often frustrated at not knowing what the possible primitive operations are [3,11], progress of the current state towards the solution, or intermediate steps to reach the solution [5]. Supporting users in decomposing complex tasks into small subtasks and incrementally composing solutions is still an open-ended research question.

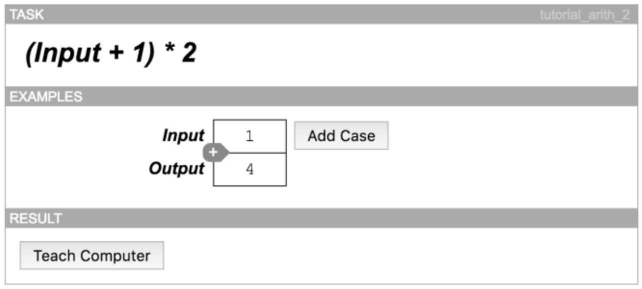
METHODS

We conducted an online user study that began with a brief introduction to PBE. Then six tutorials on the user interface and basic PBE tasks (Table 1) were given. After finishing the tutorials, participants were asked to complete five main tasks, that are advanced variations of the tutorials. Finally, the tasks were followed by a demographic survey. The study took around 26 minutes (*M* = 25.97, *STD* = 11.54), and participants who finished the entire study were paid $3.00. The study was posted on Amazon Mechanical Turk for two days, during which 161 workers started the first tutorial, 137 workers finished the tutorials and proceeded to the tasks, but only 30 finished the entire study. Summary demographics of the 30 participants who finished the entire study indicate the majority age range was 25-34 (60%, *M* = 36.43, *STD* = 7.56), male (60%), with bachelor (50%) or high school degrees (37%). The majority (84%) of participants reported that they had no programming knowledge (57%) or only basic concepts (27%). However, many of them had various IT experience, such as using spreadsheets (70%), creating web pages using HTML (30%) or content-management systems (20%), database (23%), and scripting languages such as Python or Ruby (20%).

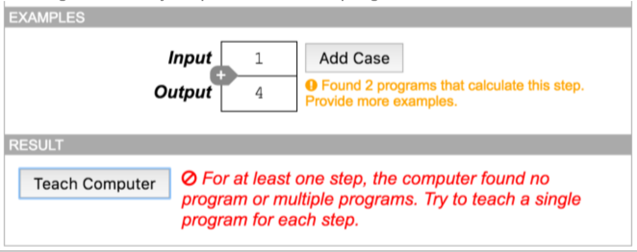
Experimental System

We developed an experimental PBE system that allows non-technical participants to quickly learn and perform decomposition and disambiguation as illustrated in Figure 1. The system can generate simple programs for standard PBE tasks (e.g. arithmetic, text processing, filtering). In the system's UI, table rows represent sequential steps from input to output, and table columns represent independent cases. Participants can type examples values in table cells, insert steps by pressing "+" buttons between rows, and add cases by pressing the "Add Case" button. Pressing "Teach Computer" runs the PBE inference engine, to generate operations that calculate each step. When the engine fails to determine operations from the provided examples, feedback

1. Initial state of the task UI that contains default Input and Output values (1 and 4), buttons for adding case ("Add Case"), adding step ("+"), and inferring operations from current examples ("Teach Computer").



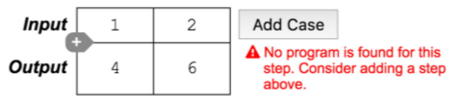
2. As the user clicks the "Teach Computer" button, the UI shows feedback messages for every step and the entire program.



Found 2 programs that calculate this step. Provide more examples.

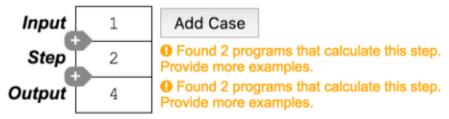
For at least one step, the computer found no program or multiple programs. Try to teach a single program for each step.

3a. As the user clicks "Add Case", an empty column is added to the right of the table in which he/she types an example (2 and 6).



No program is found for this step. Consider adding a step above.

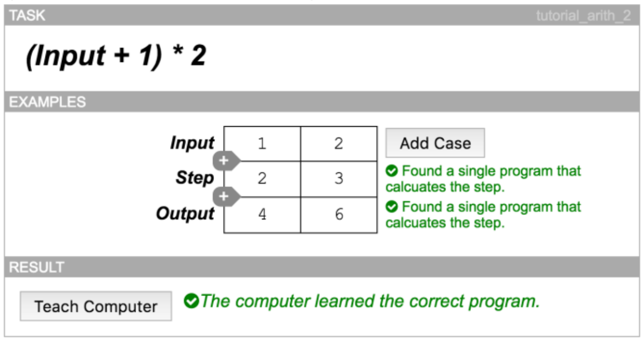
3b. Alternatively, the user could click [+] between two rows, and an extra step would be inserted between the rows.



Found 2 programs that calculate this step. Provide more examples.

Found 2 programs that calculate this step. Provide more examples.

4. By adding a case and a step, the user makes every step find a single operation, and teaches the correct operation.



Found a single program that calcuates the step.

Found a single program that calcuates the step.

The computer learned the correct program.

Figure 1. The study UI and basic walkthrough

messages are shown to the right rows. If participants spent at least three minutes, and tried (unsuccessfully) to "Teach Computer" at least eight times for a given task, a button was shown to allow them to give up and move on to the next task. Through internal pilot tests, we decided on a reasonable, high number of minutes and trials to give people the chance to try a number of answers in order to study example-providing.

		Description	Default examples		Solution examples			
Tutorials	T1, 2	Input + 1	IN	1	IN	1	5	
			OUT	2	OUT	2	6	
	T3	(Input + 1) * 2	IN	1	IN	1	2	
			OUT	4	STEP	2	3	
					OUT	4	6	
	T4	Get the sum of all numbers	IN	1,1	IN	1,1	3, 2	
			OUT	2	OUT	2	5	
	T5	Get length of a text value (including spaces).	IN	yes	IN	yes	no	
			OUT	3	OUT	3	2	
	T6	Find numbers that are greater than 9	IN	11,8,9,10	IN	11,8,9,10		
			OUT	11,10	STEP	T,F,F,T		
					OUT	11,10		
Main tasks	T7	(Input + 1) * (Input – 1)	IN	1	IN	1	2	3
			OUT	0	STEP	2	3	4
					STEP	0	1	2
					OUT	0	3	8
	T8	Sort numbers in ascending order	IN	1,-1	IN	1,-1	5,2,3	
			OUT	-1,1	OUT	-1,1	2,3,5	
	T9	Find words that are longer than two letters	IN	be, are, I, some	IN	be, are, I, some		
			OUT	are, some	STEP	2,3,1,4		
					STEP	F,T,F,T		
					OUT	are, some		
	T10	Find numbers that are not divisible by 4 without remainder	IN	1,4,5	IN	1,4,5	2,4	
			OUT	1,5	STEP	T,F,T	F,T	
					OUT	1,5	4	
	T11	Extract prices of cars that are manufactured in 2014 or later.	IN	Civic(2014)-$12000, Elantra(2012)-$9500, Corolla(2015)-$14000, Corolla(2013)-$10000	IN	Civic(2014)-$12000, Elantra(2012)-$9500, Corolla(2015)-$14000, Corolla(2013)-$10000		
			OUT	12000,14000	STEP	2014, 2012, 2015, 2013		
					STEP	12000, 9500, 14000, 10000		
					STEP	T, F, T, F		
					OUT	12000,14000		

Table 1. With the given description and default examples for each task, participants were asked to add more examples, such as the solution examples shown.

We designed two types of feedback (*simple* and *actionable*) to see whether actionable feedback effects user's behavior. The ***simple*** **feedback** provides only the number of programs that the system generated. We designed the simple feedback as the baseline condition, since most existing PBE systems [2,6,9,11] provide a similar level of feedback for generated programs. In contrast, the ***actionable*** **feedback** detects user's intentions from the examples, and explains details why it failed to generate any program and how to resolve the issue. To our knowledge, no prior PBE systems provide actionable feedback.

When the PBE system finds a **single operation** for the step, both types of feedback show the same message, "*Found a single program that calculates the step.*"

When the system finds **multiple operations**, the *simple* feedback is "*Found N programs that calculate this step*", where *N* is the number of generated operations. The *actionable* feedback is same, but adds "*Provide more examples*." to the end.

When the system finds **no operation** for the step, the *simple* feedback is "*Found no program that calculates this step.*" In contrast, the *actionable* feedback includes the following messages:

- If there is an empty cell in the current row, the *actionable* feedback is, "*There is an empty case. Did you miss filling it?*"
- If the current row contains values of multiple types (e.g. number and string), the actionable feedback is, "*There are number and string examples in this case. This might have caused the computer to fail in finding a program.*"
- If there is any row above the current row that contains all the values of the current row, the actionable feedback is, "*If you are trying to filter values from steps above, you need an additional step containing T or F.*"
- If the current row is a substring of a filtered subset of any row above, the actionable feedback is, "*Are you trying to filter and extract part of string at the same time? If that's the case, you have to do them in two steps.*"

SUCCESS RATE

As mentioned, 30 of the 161 participants finished the entire study. They successfully finished most tutorials (*average success rate* = 91.1%, *# trials* = 3.22) as shown in Table 2. The main tasks were successfully completed less often than the tutorials (*success rate* = 66.7%, *# trials* = 10.12) To understand the effect of feedback on successful task completion, we conducted a non-parametric repeated measure ANOVA test [10]. The result yielded an *F* ratio of $F(1, 150) = 26.01$, $p < .001$, indicating that the success rate was significantly greater with the actionable feedback than with the baseline feedback. We also conducted factorial ANOVA to check the effect of demographic factors on success rate, but found no significant impact ($p > .03$).

Types of Mistakes

We counted mistakes as participant errors in user-provided examples that prevent the PBE engine from generating a single program for each step. The first author reviewed 150 task results (5 main tasks done by 30 participants), and identified 246 mistakes. 25.6% of the mistakes were *critical*, meaning that they remained until participants gave up the task. We grouped mistakes into the categories below.

Missing steps (found 92 times; 30 were critical)

The PBE engine failed to generate programs when participants did not provide crucial steps as illustrated below: (a) missing steps of key values above predicates (35 times; 15 critical), (b) missing steps of predicates values above a list filtering step (31 times; 7 critical), (c) subtasks of a combination of filtering and text extraction (22 times; 15 critical), and multi-step arithmetic (T3 and T7; 4 times) as illustrated below.

(a)

IN	be, are, I, some
ST1	F,T,F,T
OUT	are, some

For T9, A predicate step (ST1) needs a step of key values ("2,3,1,4") above.

(b)

IN	1,4,5
OUT	1,5

For T10, filtered result (OUT) requires a step containing predicate values ("T" for including, "F" for excluding values).

(c)

IN	Civic(2014)-\$12000, Elantra(2012)-\$9500, Corolla(2015)-\$14000, Corolla(2013)-\$10000
STEP	2014, 2012, 2015, 2013
STEP	T, F, T, F
OUT	12000,14000

For T11, the output ("12000, 14000") is a substring of the filtered list. It requires either a substring of the original list or the filtered list above.

Ambiguous cases (29 times; 11 critical)

Participants often could not provide sufficient examples for the engine to find the right program. For example, participants stuck with single-case examples (18 times; 8 critical). (a) To generate a "not divisible by 4" condition for T10, the input requires "2", but eight participants had to try multiple times, and three of them gave up. (b) Similarly, T8 (sorting numbers) requires an additional case containing at least three numbers, whose output is not the input in reverse-order.

(a)

IN	1,4,5	2,4
STEP	T,F,T	F,T
OUT	1,5	4

For T10, to disambiguate "divisible by 4" from "divisible by 2", IN requires a value "2".

(b)

IN	1,-1	5,2,1
OUT	-1,1	1,2,5

For T8, examples for sorting must contain three numbers that are not in reverse order.

Inconsistent or unsupported values (28 times; 8 critical)

Participants provided a variety of values that the PBE engine could not find a matching program, such as inconsistent values for arithmetic tasks (9 times; 2 critical), incorrect predicates for filtering (5 times; 1 critical), and incorrectly sorted list (2 times). Participants also provided steps with single Boolean values, when the correct program requires multiple values (7 times; 3 critical). Participants often made formatting mistakes such as (a) Boolean values next to numbers (e.g. "T11, T10, F8, F9": 2 times), Boolean values without a separator (e.g. "FTFT"; 3 times) and using "Yes" and "No" instead of "T" and "F" (1 time).

		Success rate			Average # Trials (per participant)		
	Task	Base.	Exp.	χ^2 p-value	Base.	Exp.	Mann Whitney U-test
Tutorials	T1	1.00	1.00	>.5	1.67	1.07	p > .5
	T2	0.93	1.00	>.5	3.00	1.20	Z = 0.91, p < .30
	T3	0.80	0.87	>.5	6.80	3.47	Z = 2.13, p < .30
	T4	1.00	1.00	>.5	3.40	1.87	Z = 0.76, p < .30
	T5	1.00	1.00	>.5	1.20	1.07	p > .5
	T6	0.67	0.67	>.5	7.40	6.47	p > .5
Main tasks	T7	0.53	0.87	<.05	10.33	3.13	Z = 2.32, p < .01
	T8	0.67	1.00	<.03	8.73	2.73	Z = 5.56, p < .3
	T9	0.27	0.93	<.001	18.27	5.27	Z = 2.90, p < .001
	T10	0.53	0.93	<.03	13.00	4.13	Z = 1.60, p < .05
	T11	0.27	0.67	<.03	28.73	6.87	Z = 3.17, p < .001

Table 2. Success rates (proportion of participants who passed the task) and average numbers of trials for the baseline (Base.) and the experimental (Exp.) conditions. Highlighted cells are significant (p<.05).

(a)

IN	11,8,9,10
STEP	T11, T10,F8, F9
OUT	11,10

"T11" probably means that the value "11" is marked with "T"

Unnecessary steps (15 times; 5 critical)

Participants often added unnecessary steps. For example, (a) they often provided steps of unnecessary Boolean values for filtering tasks (7 times; 2 critical), numbers for arithmetic (4 times; 2 critical), or completely empty steps (2 times; 1 critical). For T10, (b) two participants provided a step that contains "4", which is the operand of the *number-predicate* program they need (2 times).

(a)

IN	11,8,9,10
STEP	T,F,F,T
STEP	T,T
OUT	11,10

The third row ("T,T") is unnecessary.

(b)

IN	1,4,5	3,8,15
STEP	4	4
STEP	F	F
OUT	1,5	3,15

To express a conditional "not divisible by 4", a participant created steps of "4" and "F".

Describing with formula (11 times; 7 critical)

Five participants described steps with formulas instead of example values. For instance, (a) they provided "Input+1", "*2", "(2)*(0)", and "+1" for arithmetic tasks (3 times; 3 critical). For the filtering tasks, they tried (b) "<2014", "1/4", "1<2<3<4", "-1<1", "are>2", and "some>2" (6 times; 3 critical). For the sorting tasks, two participants tried to describe the direction with "increasing order" and "reverse input" (2 times; 1 critical).

(a)

IN	1
STEP	Input+1
STEP	*2
OUT	4

(b)

IN	be, are, I, some
STEP	T
STEP	are>2
STEP	some>2
OUT	are, some

Inconsistent program (3 times; 2 critical)

Even when the PBE engine generated a single program for every step, the entire program could be inconsistent with the task. For instance, participants often created wrong arithmetic (2 times; 2 critical), or filtering programs (1 time).

Empty cases (2 times; 0 critical)

Participants sometimes left the right most case empty.

DISCUSSION AND FUTURE WORK

The paper presents an overview of common mistakes that non-expert users make in providing examples for PBE systems. The experiment confirms that we can automatically detect a user's programming intent, and generate actionable feedback that helps the user quickly fix mistakes. We made several simplifying assumptions that limit the scope of our findings. First, to allow non-expert users to quickly learn, the study introduces only a few standard tasks (e.g. arithmetic, string processing, and filtering). While the general patterns of findings will likely apply to other tasks, it will be important to confirm the extent to which this is true. Second, our experimental system does not show generated programs, while a few PBE systems [6,9] support interactive disambiguation where users read program descriptions and disambiguate by directly choosing a desired program. Further work is needed to explore the opportunity and effectiveness of interactive disambiguation.

The study also leads us to a wide research area. For example, how to construct and train a knowledge model of a PBE user is an open-ended research question. How various design factors effect a user's motivation and understanding of the PBE system is another research topic for future work.

REFERENCES

1. Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky (eds.). 1993. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA.
2. Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not.* 46, 1: 317–330. https://doi.org/10.1145/1925844.1926423
3. Sumit Gulwani. 2016. Programming by Examples (and its applications in Data Wrangling). In *Verification and Synthesis of Correct and Secure Systems*, Javier Esparza, Orna Grumberg and Salomon Sickert (eds.). IOS Press.
4. Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, and Benjamin Zorn. 2015. Inductive Programming Meets the Real World. *Commun. ACM* 58, 11: 90–99. https://doi.org/10.1145/2736282
5. Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In (UIST '11), 65–74. https://doi.org/10.1145/2047196.2047205
6. Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In (CHI '11), 3363–3372. https://doi.org/10.1145/1978942.1979444
7. Tessa Lau. 2009. Why PBD systems fail: Lessons learned for usable AI. *AI Magazine 30.4*, 65.
8. Henry Lieberman. 2001. *Your Wish is My Command: Programming By Example*. Morgan Kaufmann, San Francisco.
9. Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. 2015. User Interaction Models for Disambiguation in Programming by Example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software &#38; Technology* (UIST '15), 291–301. https://doi.org/10.1145/2807442.2807459
10. Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011. The Aligned Rank Transform for Nonparametric Factorial Analyses Using Only Anova Procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11), 143–146. https://doi.org/10.1145/1978942.1978963
11. Kuat Yessenov, Shubham Tulsiani, Aditya Menon, Robert C. Miller, Sumit Gulwani, Butler Lampson, and Adam Kalai. 2013. A Colorful Approach to Text Processing by Example. In (UIST '13), 495–504. https://doi.org/10.1145/2501988.2502040