Human language and its role in reference-point errors

Craig S. Miller
School of Computing
DePaul University
cmiller@cdm.depaul.edu

July 30, 2016

Abstract

A reference-point error occurs when a programmer writes code that mistakenly refers to one element when the intention is to refer to an element structurally related to it. I review these errors and their relation to the use of metonymy in human communication. Using a working example, I draw upon cognitive theories of human communication and problem-solving to explore three accounts of why these reference errors occur in novice programming. The first account involves a deficient mental model, the second assumes a misconception of the notional machine, and the third considers implicit, proceduralized habits of communication. I conclude with learning objectives for students that address these sources of difficulty.

1. Introduction

Human language has long served as a basis for analyzing the errors and misconceptions of novices as they learn to program (e.g. Clancy, 2004; Bonar & Soloway, 1985; L. A. Miller, 1981). In many cases, novice mistakes involve conflating the human language meaning of a word with its meaning in the programming context. For example, Spohrer and Soloway (1986) discuss how the English meaning of the word 'OR' may be misapplied in the construction of a Boolean expression. Other researchers have taken a broader view. For example, Tenenberg and Kolikant (2014) discuss how practices of human communication in general may shape novices approach to programming. Despite these efforts, less understood is the underlying cognitive mechanisms for which human language may shape novice performance when learning to program. There is some uncertainty as to whether language knowledge is a cause of the mistakes or whether it too is just a reflection of underlying mental representations.

The goal of this paper is to explore possible mechanistic relationships between human language and novice programming behavior. It does so with a pointed focus on just one aspect of programming, namely that of reference specification. Moreover, it just focuses on one type of error, albeit one that has been repeatedly identified and then systematically studied. Here the novice behavior involves reference-point errors and their parallel to metonymy in human communication. Metonymy in human language occurs when the speaker explicitly names one element with the intention of referencing a related element (see Lakoff & Johnson, 1980, for an overview of metonymy in the context of other figurative language). In previous work I have presented how reference-point phenomena in novice programming matches those in human communication (C. S. Miller, 2014) but have yet to explore possible sources of the phenomena in detail.

In this paper, I draw upon cognitive theories of human communication and problem-solving to suggest plausible accounts of why reference errors occur in novice programming. Of course, reference errors that parallel the use of metonymy only represent a very small portion of novice difficulties as they learn to program. Yet, reference errors have been identified as a significant source of difficulty for students (Goldman et al., 2008). Perhaps more importantly, the clear focus allows us to explore possible theories in greater detail and work us toward a more coherent and systematic theory. If successful, it may provide some direction for analyzing other categories of novice errors.

2. Metonymy and Programming

A reference-point error is an expression that refers to an entity that is not the programmer's intended target. Often the intended referent and the actual referent are elements in a computer data structure, but

they could also be file references, database elements or conceptual constructions. While all programmers may make reference-point errors, scholars have noted that novice programmers have particular difficulty with them. For example, Du Boulay (1986) describes "confusion between the subscript of an array cell and the value stored there." Holland, Griffiths, and Woodman (1997) describe cases where students conflate a whole object and an identifying attribute of the object. In these cases, a student may write a coded reference that indicates an attribute when the specification required a reference to the object itself. Reference-point errors have also been noted in exercises where students are asked to give precise instructions to the instructor acting as a robot (Davis & Rebelsky, 2007).

In previous work, I analyzed reference-point errors in terms of metonymy (C. S. Miller, 2014). Metonymy is a rhetorical device used in human-to-human communication, often to emphasize a particular attribute or to aid identification. For example, consider the phrase: "Open the ice cream and serve two scoops." In this phrase, the intention is not to literally open the ice cream but rather the container holding the ice cream. The speaker may be emphasizing the ice cream in order to distinguish it from sorbet or some other frozen dessert. The container does not need to be mentioned since the recipient of the request can readily infer that it is the container of the ice cream that needs to be opened.

Use of metonymy in human language parallels some reference-point errors produced by novice programmers. Consistent with the observations by Holland et al. (1997), students are more likely to specify an attribute in place of an object (and vice versa), when the attribute is an identifying property such as the name. Elsewhere I have presented multiple experiments that systematically produce this effect and discussed its relationship with metonymy (C. S. Miller, 2012, 2014). Finally, other accounts (Ragonis & Ben-Ari, 2005; Vahrenhold & Paul, 2014) note student confusion between an object's identity and its attributes, although these observations are not framed in terms of human-to-human communication.

The goal of this paper is not to provide a comprehensive account of reference-point errors, nor even an account of all such errors based on metonymy. Rather, we use this kind of reference error as a particular case that has a clear specification, directly parallels metonymic constructions in human language and is documented in diverse reports. This direction does not just consider cognitive mechanisms but allows us to explore external factors that shape mental models and habits of communication, considerations advocated by Tenenberg and Knobelsdorf (2014). Similarly use of language may influence how students construct their mental models (Holmboe, 2005; Diethelm & Goschler, 2015).

2.1. Working Example

In this section, I will provide an example for further reference throughout the paper. While the syntax has been simplified to make it more analogous to diverse programming contexts, the construction is derived from previous study (C. S. Miller, 2014). The working example will also illustrate the phenomena presented in this study (also consistent with other reports Davis & Rebelsky, 2007; Holland et al., 1997).

The example assumes a set of objects (described by attributes and values) and an API for manipulating them. This API provides a simple retrieval of an object based on an attribute and value of the object. The retrieved object can then be added to a collection object. Below is an example of its correct use:

```
obj = catalog.find("name", "peach")
cart.add(obj)
```

Note that the retrieval requires the explicit specification of the attribute (i.e. "name") and its value (i.e. "peach"). The retrieved object can then be passed to the **add** method as a parameter. For now, we assume that the API has been reasonably presented to students, although, as we shall see, as students commit a reference-point error, they are essentially incorrectly using the API.

While the API is not the programming language, it offers instructions that students need to understand for successful programming. In this context, the API arguably extends what Du Boulay (1986) calls the notional machine. As Sorva (2013) explains, the notional machine includes what students need to learn for successful programming in a particular context. Later in this section we will see an example that does not require an API for eliciting similar reference errors.

Given the API, a reference-point error occurs when the student tries to add the item by using one of its attributes (e.g. name) instead of the object itself. Below is such an example:

```
cart.add("peach")
```

In another version of the error, a student may try to access the object using the attribute as a principal identifier and thus omit the explicit reference to the attribute name. In this example, the student may be using the string "peach" as the fundamental means for accessing the object rather than realizing that "peach" is just another attribute value for the object.

```
obj = catalog.find("peach")
cart.add(obj)
```

As already discussed, students are more likely to commit these reference-point errors when the attribute is an identifying property (e.g. name, title, label) instead of a descriptive property (e.g. color, texture) as correctly demonstrated below:

```
obj = catalog.find("color", "yellow")
cart.add(obj)
```

Note that this initial presentation of the working example does not provide the whole context for which it would be applied. For example, we have (so far) not considered how students are taught that the *add* operation requires an object reference. Also, the task instructions may bias the student towards a reference-point error. Let us consider the following task instructions:

1. Add the peach to the cart
2. Add the peach object to the cart
3. Add the object whose name is peach to the cart

The first of these task instructions is effectively using metonymy while the remaining instructions indicate a distinction between the object and the attribute. Presumably students would be more likely to commit a reference-point error with the first of these task instructions. However, previous work reports students making these mistakes even when the task instructions explicitly indicated the attribute and its value such as the wording found in the last task instruction (C. S. Miller, 2014).

An alternate version of the task may ask students to write a general function for adding by name any object to the cart. Below is a correctly coded example:

```
def addToCard(item, cart)
  obj = catalog.find("name", item)
  cart.add(obj)
end
```

Different contexts such as this task may produce different results and our analysis should provide some insights on how student answers may vary depending on the context.

While the working example in this paper uses an API, reference difficulties may occur without an API. Below is an example, where the task is to write a function that serially searches an array of objects, returning true if there is an object that matches its name attribute:

```
def isNameInList(item, list)
  list.each do |obj|
    if obj.name == item
      return true
    end
  end
```

```
    return false
  end
```

This next example shows this example with a reference-point error consistent with the use of metonymy:

```
def isNameInList(item, list)
  list.each do |obj|
    if obj == item
      return true
    end
  end
  return false
end
```

In this example, the intended referent is the **name** attribute for the **obj** object, but the code refers to the whole object in its place. While reference-point errors have yet to be studied extensively in this non-API context, a forthcoming paper (C. S. Miller & Settle, 2016) reveals similar reference-point errors including those consistent with previous findings.

3. Overview of theories

The source and even the definition of metonymy in human communication is subject to some debate. For example, some scholars question whether metonymy represents a shift in reference or simply involves a shift in meaning (Rebollar, 2015). More generally, such as in the construction of noun phrases, there are opposing views as to whether speakers deliberately construct a reference to aid the recipient in efficiently identifying the referent or whether its construction is more a product of what is active in the speaker's working memory (Gatt, Krahmer, van Deemter, & van Gompel, 2014).

For this paper, the goal is not so much to resolve these debates but to explore possible sources for the errors seen in programming. I nevertheless draw upon the diverse theories of metonymy in human language and possible mechanisms that underlie it. Such treatment will allow us to consider how these mechanisms might relate to the reference-point errors we see in novice programming. These mechanisms thus hypothesize one source or even a combination of sources when a student constructs an incorrect reference.

As we look at sources, we will look beyond the cognitive mechanisms employed by the student, and include background knowledge and context, which arguably inform student behavior (Tenenberg & Knobelsdorf, 2014). As we shall see, all draw upon external domain knowledge that is arguably involved in creating and resolving metonymic references for human communication (Croft, 1993).

3.1. Mental Models and Verbal Reasoning

We first consider how a student's mental model of an object may lead to reference-point constructions that parallel the use of metonymy. A mental model is person's conceptualization of how a system or artifact works in the environment. Norman (1983) has shown how differences between a human user's conceptual model and an accurate working model can account for errors in the context of human-machine interaction. Similarly, Sorva (2013) discusses how differences between a student's mental model of the notional machine and an effective conceptualization of it can account for novice programming errors.

In this paper, I propose possible differences between a student's model and an effective model in order to account for reference errors. I draw upon mental model theory as it has been applied to deductive reasoning in the form of syllogisms (Johnson-Laird, 1986; Polk & Newell, 1995). In these problems, people are given premises (e.g. "All clowns are artists", "no mechanics are artists") and asked to produce a valid conclusion (e.g. "no mechanics are clowns").

Johnson-Laird (1986) theorized that people work with situation models and their ability to produce valid conclusions depends on their ability to reason with multiple alternative models. Failure to consider alternative models accounts for many of the errors observed in the studies. In a departure from the

Deficient model

```
peach | color: yellow, size: large, shape: sphere
apple | color: green, size: medium, shape: square
pear | color: green, size: large, shape: pear

add peach to cart
```

Effective model

```
obj1 | name: peach color: yellow, size: large, shape: sphere
obj2 | name: apple color: green, size: medium, shape: square
obj3 | name: pear color: green, size: large, shape: pear

add obj1 to cart
```

Figure 1 – Two mental models encoding the task of adding an object.

Johnson-Laird account, Polk and Newell (1995) hypothesize that people construct situation models as an immediate product of processing linguistic input. Characterized as "verbal reasoning," the ability to draw valid conclusions depends on successfully encoding and reencoding the premises in an annotated situation model. Both the mental model theory of Johnson-Laird and the verbal reasoning theory make use of situation models instead of logical inference rules to account for human performance.

A consequence of verbal reasoning is that people often make assumptions that are not warranted or produce models that are incomplete. Moreover, their structure may facilitate particular inferences or operations while omitting other possibilities, unless further encoding or reformulation occurs. Since people have well practiced routines constructing mental models as they comprehend linguistic content, they are likely to apply such routines to diverse domains. Here I explore how a plausible encoding of a computer programming task could lead to the reference-point errors presented in our working example.

In the style of Johnson-Laird's models, Figure 1 depicts two mental models that encode the example problem from the previous section. They include both the goal (i.e. "add the peach to the cart") and a model of various objects, one of which is the targeted object. The first (deficient) model would produce a reference-point error by referring to the object by its name rather than the object itself. The second (effective) model is a possible reencoding, which properly distinguishes between the name of the object and a reference that refers to the whole object.

Let us consider the first, deficient model in more detail. While the actual details would vary among students and contexts, application of the model could lead to a reference-point error if it has the following characteristics:

1. Each object consists of a list of values (e.g. yellow, large), each with a named attribute.
2. Each element has a singular value that primarily identifies the object from a set of objects.
3. The identifying value does not have an explicitly named attribute.

Technically the identifying value (e.g. 'peach') is just another attribute of the model, but its privileged representation could lead to a reference error. If students applied this model, they may write code that just references attribute and not the whole object:

```
cart.add("peach")
```

Following Polk and Newell's verbal reasoning hypothesis, the construction of this deficient mental model is likely a consequence of human language and communication. First, the name of the attribute is significant. Knowing its identifying role gives it a privileged state as reflected in the model. Second,

the model may have also been constructed by presenting objects using metonymic constructions. The following examples use metonymic constructions to reference objects in the domain:

1. In this example, we will add a peach to the cart.

2. This literal representation of the pear object shows that it has the attributes of color, size and shape.

3. Note that the peach and the pear have the same size.

In all cases, the name attribute is used to refer to the object without explicitly indicating the attribute. A literal interpretation would then lead to the construction of the first (deficient) model show in Figure 1. More generally, because we routinely use metonymy to describe our world, this language may produce mental models that reflect the literal interpretations of task instructions. Reference-point errors result as a straight-forward application of literal interpretation.

Of course a student may construct a more accurate model that correctly distinguishes between the identifying attribute and the object itself. Such is the 'Effective model' in Figure 1. This model more accurately represents how objects are actually constructed. From this construction, a student could reason that the name is one of several attributes that does not technically identify the needed object for the operation of adding to the cart (although an API could be designed and implemented to allow such an operation). The presentation of this particular model commits to naming the connecting symbols (i.e. 'obj1', 'obj2' and 'obj3'), although this detail may be unspecified in a hypothesized working model.

3.2. Communication design

Another possibility is that the student knows that the name is organized by attribute but believes that the computer can resolve a reference that omits the relationship between the object and the identifying attribute value. In this case, the student's mental model may explicitly distinguish between the name attribute and the reference to the entire object. Yet, the student knowingly refers to just the name value with the view that the identifying nature of its name allows the computer to effectively infer that the whole object is intended.

Here the novice programmer may be intentionally employing principles of communication such as those presented by Grice (1975). Perhaps most relevant is Grice's maxim on Quantity, where the amount of information should be no more or less than what the recipient requires. If the student believes that the system can effectively resolve the reference without explicit mention of the attribute, then it is a reasonable communication principle to omit the attribute. Moreover, for human communication, the so-called literal expression is not necessarily more efficient than an expression based on metonymy. It has been well argued that the literal interpretation is not necessarily processed first by human listeners (Recanati, 1995).

Knowingly referencing the attribute in place of the whole object suggests that the student believes that some provision has been made so that the computer can successfully resolve the attribute reference to the object itself. Pea (1986) calls this type of mistake a "hidden mind superbug" if the student performs as if the computer has a human-like mind that can successfully infer the intention of the student programmer. Pea warns that students do not necessarily believe that the computer literally works like a human mind. In fact, if asked, the student may disavow a hidden mind. For this reason, Pea suggests that this bug may be a product of unconscious knowledge, something we consider in the next section.

The explicit assumption that the computer can infer the intended referent may seem naive, unless we consider that designers of computing technology often consider the likely intentions of its users or programmers. In this specific case, it would not be unreasonable to provide a method in the API that gives identifying attributes, such as name, a privileged method for referring to objects when adding them to the cart. In any case, as Sorva (2013, p. 8:7) notes, "the novice needs to learn what the notional machine does for them on one hand, and what their responsibility as a programmer is on the other."

```
  object: <label>
  goal: add to <cart-obj>
--->
  write-code <cart-obj>.add(<label>)
```

Figure 2 – Production rule for referencing an object.

3.3. Acquired habits of communication

We finally consider the case where a student relies on implicit communication knowledge for constructing the reference. In this case, the student may possess an effective mental model (such as that presented in Figure 1) and have no (explicit) expectation that the computer can successfully resolve the point of reference. Instead, the reference error occurs by drawing upon implicit, procedural skills, plausibly obtained through the practice of human-to-human communication. At one time, the source of this practice may have been the same as used to produce Grice's maxims (Grice, 1975) or processes for selecting attributes, such the computationally efficient method proposed by Dale and Reiter (1995). In such cases, the student does not need to be aware that he or she is drawing on these practiced skills.

A proceduralized application of knowledge can be modeled with a production-based system. Productions are associative rules that match internal representations (including those that correspond to mental models) and perform an action, either to the internal representations or as a perceptual/motor operation. Production systems have been offered as cognitive architectures, such as Soar (Newell, 1994) and ACT-R (Anderson, 1993). In these models, productions can be acquired through deliberate practice. Once acquired, explicit access may not be available.

Here I consider a simple production-based model that would produce a reference-point error. Figure 2 depicts a production that selects an attribute from an object and uses it to write code. The elements in brackets (e.g. <label>) are variables and can match any symbol. Its condition matches one object with any attribute and the goal of adding that object to the cart. The production is very general and could match in many ways. However, assuming an activation-based process for selecting a production, the production only matches the attribute with the greatest activation and directly places it in the expression.

If the model assumes that identifying attributes are most salient, carrying the greatest activation, it accounts for why reference-point errors occur more frequently with identifying attributes than those that are just descriptive. In a model, salience could be based on perceptual properties as well as any preprocessing that highlights the attribute. Such preprocessing may draw upon attribute selection algorithms such as those proposed by Dale and Reiter (1995) in human-to-human communication.

This third account does not depend on a particular mental model—the production could match either representation in Figure 1. It also does not assume any explicit understanding of the notional machine. Instead, it relies on default habits, plausibly acquired or at least reinforced in human-to-human communication.

4. Discussion

The three accounts of reference-point errors stem from three different sources: a deficient mental representation, a misunderstanding of what the system can do, and a reliance on implicit habits for communication. Since these sources are not mutually exclusive, it is possible that student mistakes arise from any of them, given the right context.

A goal for future work is the development of strategies for diagnosing which account is producing a reference error for any given circumstance. For example, it may be possible to manipulate task instructions to encourage the construction of an effective mental model. If a defective mental model is a root cause, better instructions should reduce the frequency of errors. As another example, priming selected elements may elicit different habits and thus show support for the third account.

In the absence of an effective strategy for identifying the source of errors, the three accounts nevertheless

suggest necessary prerequisites for successfully producing a well-formed reference. These accounts allow us to devise learning objectives for students. For successfully constructing a reference, students need to learn the following:

- Fully encode attribute/value pairs for modeling an object.
- Acquire a rigorous model of the notional machine.
- Obtain practiced routines (i.e. productions) for extracting the appropriate elements from the mental representation.
- Employ a validation step to ensure proper alignment.

The first three correspond to each of the three accounts. Fully encoding the object in terms of its attributes and values gives students explicit access to the identifying attribute name. Understanding the limits of the notional machine—the API in our example—informs their obligation to fully specify the reference. Obtaining a practiced routine ensures proper inclusion of the needed attribute label and increases its likelihood to be selected among other competing habits. This practice also enables students to appropriately chunk components to make better use of working memory (Soloway & Ehrlich, 1984). Finally, the last learning objective could be deployed as a defense against any of the error accounts.

Experimenting with instructional interventions that focus on any particular objective may also provide indirect evidence of where students have most difficulty. For example, an exercise may ask students to draw representations of the objects and check them against correct answers. If such an exercise improves student performance on specifying references, it suggests that the underlying mental model had been at fault.

Another issue for further work is whether these accounts are relevant for other disciplines. As an example, Zandieh and Knapp (2006) discuss the role of metonymy in mathematical understanding. They report that a student may express a derivative as the tangent line rather than the slope of the tangent line. Like the analysis here, instructional materials may have led students to develop a deficient mental model that ultimately affects their ability to solve problems. On the other hand, unlike almost any other discipline, problem solving in computing requires human-to-machine communication, an activity whose difference from the human-to-human kind can cause its own problems.

5. References

Anderson, J. R. (1993). *Rules of the mind*. New York: Lawrence Erlbaum Associates.

Bonar, J., & Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction*, *1*(2), 133–161.

Clancy, M. (2004). Misconceptions and attitudes that interfere with learning to program. In S. Fincher & M. Petre (Eds.), *Computer science education research* (pp. 85–100). Taylor and Francis Group, London.

Croft, W. (1993). The role of domains in the interpretation of metaphors and metonymies. *Cognitive Linguistics*, *4*(4), 335–370.

Dale, R., & Reiter, E. (1995). Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive science*, *19*(2), 233–263.

Davis, J., & Rebelsky, S. A. (2007). Food-first computer science: starting the first course right with PB&J. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on computer science education* (pp. 372–376). New York, NY, USA: ACM.

Diethelm, I., & Goschler, J. (2015). Questions on spoken language and terminology for teaching computer science. In *Proceedings of the 2015 acm conference on innovation and technology in computer science education* (pp. 21–26).

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

Gatt, A., Krahmer, E., van Deemter, K., & van Gompel, R. P. (2014). Models and empirical data for the production of referring expressions. *Language, Cognition and Neuroscience*, *29*(8), 899–911.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a delphi process. In *Proceedings of the 39th sigcse technical symposium on computer science education* (pp. 256–260). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1352135.1352226` doi: 10.1145/1352135.1352226

Grice, H. P. (1975). Logic and conversation. In P. Cole & J. L. Morgan (Eds.), *Syntax and semantics volume 3: Speech acts.* Academic Press, New York.

Holland, S., Griffiths, R., & Woodman, M. (1997). Avoiding object misconceptions. *SIGCSE Bull.*, *29*(1), 131–134.

Holmboe, C. (2005). Conceptualization and labelling as cognitive challenges for students of data modelling. *Computer Science Education*, *15*(2), 143–161.

Johnson-Laird, P. N. (1986). *Mental models*. Cambridge, MA, USA: Harvard University Press.

Lakoff, G., & Johnson, M. (1980). *Metaphors we live by*. Chicago, IL: The University of Chicago Press.

Miller, C. S. (2012). Metonymic errors in a web development course. In *Proceedings of the 13th annual conference on information technology education* (pp. 65–70). New York, NY, USA: ACM.

Miller, C. S. (2014). Metonymy and reference-point errors in novice programming. *Computer Science Education*, *24*(3).

Miller, C. S., & Settle, A. (2016). Some trouble with transparency: An analysis of student errors with object-oriented python. In *Proceedings of the 12th annual conference on international computing education research.* New York, NY, USA: ACM. Retrieved from `http://dx.doi.org/10.1145/2960310.2960327` doi: 10.1145/2960310.2960327

Miller, L. A. (1981). Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, *20*(2), 184–215.

Newell, A. (1994). *Unified theories of cognition*. Harvard University Press.

Norman, D. A. (1983). Some observations on mental models. In D. Genter & A. L. Stevens (Eds.), *Mental models* (pp. 7–14). Psychology Press, New York.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, *2*(1), 25–36.

Polk, T. A., & Newell, A. (1995). Deduction as verbal reasoning. *Psychological Review*, *102*(3), 533.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of oop concepts by novices. *Computer Science Education*, *15*(3), 203–221.

Rebollar, B. E. (2015). A relevance-theoretic perspective on metonymy. *Procedia-Social and Behavioral Sciences*, *173*, 191–198.

Recanati, F. (1995). The alleged priority of literal interpretation. *Cognitive science*, *19*(2), 207–232.

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programming knowledge. *Software Engineering, IEEE Transactions on*(5), 595–609.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(2), 8.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, *29*(7), 624–632.

Tenenberg, J., & Knobelsdorf, M. (2014). Out of our minds: a review of sociocultural cognition theory. *Computer Science Education*, *24*(1), 1–24.

Tenenberg, J., & Kolikant, Y. B.-D. (2014). Computer programs, dialogicality, and intentionality. In *Proceedings of the tenth annual conference on international computing education research* (pp. 99–106). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2632320.2632351` doi: 10.1145/2632320.2632351

Vahrenhold, J., & Paul, W. (2014). Developing and validating test items for first-year computer science courses. *Computer Science Education*, *24*(4), 304–333.

Zandieh, M. J., & Knapp, J. (2006). Exploring the role of metonymy in mathematical understanding and reasoning: The concept of derivative as an example. *The Journal of Mathematical Behavior*, *25*(1), 1–17.