What's in a Linked List? A Phenomenographic Study of Data Structures Diagrams

Anonymous Author(s)

ABSTRACT

Diagrams in data structures provide a valuable context for untangling the relationship between spatial ability and persistence in computer science. Spatial ability is a strong predictor of success in computer science, and data structures rely on spatially oriented language and tasks (e.g., *rotating* binary trees, *collisions* in hash tables). While we know that spatial ability is important for succeeding in computer science, we have little understanding about why spatial ability is important for succeeding in computer science. In this paper, we present an initial study using phenomenographic methods to explore how YouTubers draw and animate linked list diagrams in instructional videos. Through inductive coding, we developed a code book to describe how the diagrams were crafted. While YouTubers used consistent language (e.g., "head," "tail," "node") and every YouTuber used diagrams, there was considerable variance in how linked lists were represented. Representational choices seemed to change in response to instructional goals or tasks.

CCS CONCEPTS

• **Theory of computation** → **Data structures design and analysis**; • **Human-centered computing** → *Visualization techniques*; • **Social and professional topics** → **Computer science education**.

KEYWORDS

data structures, linked lists, diagrams, spatial ability

ACM Reference Format:

Anonymous Author(s). 2018. What's in a Linked List? A Phenomenographic Study of Data Structures Diagrams. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Spatial ability strongly correlates with and predicts success in early Computer Science (CS) [5, 11, 26, 37], but little is known about why. Data structures are common topics in CS1/CS2 curricula and have semi-canonical diagrams that may be spatially oriented (e.g., nodes *moving*, pointers *pointing*). Linked lists are considered a required part of CS curricula [2], are taught early to novices in CS1/CS2, and are a building block for more complex data structures (e.g. binary search trees and hash tables). Thus, studying linked lists may provide an avenue to explore the relationship between CS content and spatial ability. However, to the best of our knowledge, there is no formal documentation on how instructors represent linked lists or how those representations are used. Much of the existing literature focuses on visualizers (e.g., Online Python Tutor [18], CSTutor [7]), and these visualizers impose diagram standards onto their users.

Informally, we have seen a variety of diagrams for illustrating linked lists that vary in what information is made explicit or kept implicit. Likewise, in conversations with colleagues, we have had many discussions about our perceptions of the affordances and drawbacks for different styles of diagrams. We are not aware of any studies that have sought to systematically document the parameter space for these diagrams and how that parameter space is used. By documenting the parameter space, we can more formally interrogate whether different styles of diagrams affect student learning, particularly as it pertains to spatial ability and the science of diagrams and reasoning. We take our first steps toward creating this foundation for future research by first analyzing freely available instructional videos on linked lists from YouTube™and by asking the following research questions:

RQ1: What is the parameter space of diagrams YouTubers use when discussing linked lists?

RQ2: How do YouTubers change these diagrams when used to illustrate algorithms?

2 LITERATURE REVIEW

2.1 Diagrams and Reasoning

Diagrams can help offload cognition [50] and are frequently used to reason about abstract data structures. Using diagrams leads to a variety of common metaphors and imagery that are visually easier to reason about than code or low-level memory models (e.g., an array is a *contiguous set of boxes*; hashes separate things into *buckets*; a *bushy* binary search tree is better than a *spindly* one). Thus, instructors and students alike often sketch or visualize diagrams when designing algorithms, leading to spatially oriented tasks (e.g., *splitting* unsorted arrays, *merging* sorted arrays, *preventing collisions* in hash tables, or *rotating* binary trees). However, this presents a natural problem for low spatial ability students who may have difficulty visualizing diagrams and their manipulations.

For a diagram to be useful, the diagram needs to make important concepts explicit in its features and be clear in what its features can or cannot do. As students progress and gain more knowledge, their domain expertise helps guide their attention and fill in implicit information. Hegarty [20] argues that when presented with instructional diagrams, a novice's attention is focused on perceptually salient diagram features (e.g., high contrast shapes or colors). Unfortunately, what a novice might consider perceptually salient may not be conceptually important, and what an instructor may consider conceptually important may not even be present. Hegarty,

Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Canham, and Fabrikant [21] studied perceptual salience in weather maps and found that participants were more accurate if they viewed maps that made task-relevant information more visually salient. Johnson-Glauch, Choi, and Herman [25] found that students failed to use knowledge they possessed when there was not an explicit feature in the diagram indicating to use that knowledge. The design of diagrams can also hinder learning. Johnson-Glauch, et al. [25] also found that students conflated different concepts when both represented arrows. Similarly, Heiser and Tversky [23] found that arrows in mechanical diagrams can communicate multiple meanings (sometimes simultaneously) such as sequential steps or path of motion. Arrows in linked data structure diagrams may also have multiple meanings depending on how they are used.

Despite the prevalence of diagrams in a data structures context and the existing literature on diagram design, we lack an understanding of what kinds of diagrams are used in the classroom. Mazumder, Latulipe, and Pérez-Quiñones [31] studied variable, array, and object diagrams in 15 Java textbooks and found that many diagrams were not explanative in terms of system topology or behavior. However, they did not explicitly document the variance of the diagrams found. Chotzen, Johnson, and Desai [8] found that while students appear to understand linked list diagrams, they had difficulty understanding pointer reassignment when applied to insertion or deletion algorithms. Thus, it is difficult to quantify which diagrams are helpful, effective, or spatially oriented when we do not understand the parameter space for constructing diagrams.

2.2 Spatial Ability

According to Margulieux [30], spatial reasoning "is the mental processing of spatial, non-verbal information" (p. 82), such as mental rotations and translations of objects. Spatial ability and spatial skill refer to an individual's upper limit and current level of spatial reasoning, respectively. In this paper, we will use spatial ability to refer to both.

There is growing interest in spatial ability and its predictive power on students' success in computing. Spatial ability has been correlated with success in a number of STEM fields, and appears most important for novices [48]. In a 50-year longitudinal study, Wai, Lubinski, and Benbow [49] showed spatial ability was a key predictor of STEM achievement and success, and others have shown similar correlations specifically in computer science [5, 11, 26, 37]. Unfortunately, spatial ability becomes a gatekeeper to success because novices must grapple with spatial ability in early coursework, whereas experts have built up the domain knowledge and expertise to cope with problem-solving without relying on purely spatial means [45, 48, 49], similar to how experts can fill in implicit information in diagrams. Further complicating the picture, studies have shown a gender gap in spatial ability favoring men [29] and a socioeconomic gap favoring the wealthy [36].

To cope with these issues, researchers have looked to interventions to bridge these gaps. In chemistry, Stieff, et. al [44] found that training students in mixed spatial-analytic problem-solving strategies eliminated the gender gap in an introductory organic chemistry series. However, Stieff's interventions depended on the invariant, canonical nature of diagrams in chemistry education, whereas computer science instructors do not agree on canonical diagram conventions or how existing diagram conventions should be used. In engineering, Sorby and colleagues developed a spatial ability course for low spatial ability students, resulting in significantly improved grades in later courses and better retention rates [42, 43]. In computer science, there have now been a few studies demonstrating that spatial ability training can help students succeed in CS1 [6, 11, 38].

However, few studies try to tease apart the relationship between general spatial ability and less obviously spatial, domain-specific content. In mathematics education, Hegarty and Kozhevnikov [22] showed the relationship between spatial ability and problem solving strategy when applied to word problems: students who identified spatial relationships in the problem and drew diagrams accordingly performed better than students who did not.

3 METHODS

We used a phenomenographic approach to capture as many diagram features as possible. Phenomenography seeks to describe the variety of ways a group experiences and thinks about a topic. In our case, we are interested in how a community of YouTubers conceptualizes linked lists. Because of their prevalence in CS1/CS2-type courses (and in programming interview question pools), we chose to focus on linked lists and expected to see a variety of diagrams. A key advantage of linked lists is that insertion is fast relative to arrays, so we expected many YouTubers to mention insertion as a main motivation for using them and to explain insertion algorithms. We describe the data collection and analysis processes below.

3.1 Data Collection

To explore the parameter space of linked list diagrams, we searched YouTube™ for video resources. YouTubers, particularly independent creators, come from all over the world and likely feel there is a gap in existing videos that they can fill. This leads to a high degree of variability, maximizing our data sampling. YouTube™ provides a rich resource pool for our initial study, which will enable more targeted data collection in the future. From the student perspective, these free videos are helpful supplementary materials with thousands (sometimes over a million) views and generally positive comments (top comments frequently mention that the YouTuber was better than their university professors).

To gather videos for analysis, we followed a literature review-style approach: we used an Incognito tab on Google Chrome to avoid search bias, agreed on keywords to use as search terms, and pulled as many videos that used those keywords in the title of the video. To look for generic videos covering singly linked lists, we searched for "linked list." From this initial search, we found 13 generic videos covering linked lists: Video #1 [16], Video #2 [19], Video #3 [10], Video #4 [4], Video #5 [24], Video #6 [17], Video #7 [32], Video #8 [14], Video #9 [46], Video #10 [33], Video #11 [3], Video #12 [39], and Video #13 [41]. From the 13 generic videos, we decided to exclude Video #11 because the YouTuber anthropomorphized nodes as people and described these nodes in a way that none of the coders could map consistently to more standard linked list diagrams or conventions. Additionally, many commenters also mentioned being confused.

After initial coding of the 13 generic videos (see Section 3.2), the last author found 5 additional generic videos to help verify codes and resolve disagreements: Video #14 [40], Video #15 [34], Video #16 [12], Video #17 [9], and Video #18 [15].

To look for diagrams in action during insertion, we searched for "linked list insertion," and found 4 videos specific to insertion: Video #19 [13], Video #20 [35], Video #21 [28], and Video #22 [47]. Additionally, some of the previous generic videos included clips of insertion, so these were added for analysis. For insertion-specific videos and clips, we wanted to keep the focus on diagrams, so we only included clips of diagram manipulation on a non-empty list example. For YouTubers who used a diagram when illustrating insertion but implemented the algorithm in code separately, we included a clip of their coding process for comparison. See Table 1 for a summary of videos found.

Video Category	Search Term	# Of Videos
Generic	"linked list"	18
Insertion-specific	"linked list insertion"	4
Prepend Clips	N/A	7
Insert Anywhere Clips	N/A	10
Append Clips	N/A	5

Table 1: Categories of videos curated from YouTube™. Note: a mixture of Generic and Insertion-specific videos contributed to the Prepend, Insert Anywhere, and Append clips.

As shown above, every video has been cited in the references, but we will refer to the videos by a random identifier (e.g., Video #1, Video #2, Video #3) to avoid interpreting our analysis as an evaluation or critique of individual YouTubers.

3.2 Qualitative Methods

To answer **RQ1**, we performed inductive coding on the 13 generic videos from YouTube™. Each author coded the videos separately, then came together and developed a code book. We use Krippendorff's α to measure the inter-rater reliability of our code book [27]. The code book and four, randomly-selected videos were sent to an outside colleague to check the clarity of the code book and the ability of others to apply our code book. This first external check did not yield satisfactory reliability ($\alpha = 0.42$), so we revised and clarified the code book. The authors then sent the updated code book and the same four videos to a second colleague, reaching satisfactory reliability ($\alpha = 0.77$). After a second round of refinement, the authors used the same code book to code the 5 additional generic videos found by the last author to check our internal reliability. We calculated Krippendorff's alpha $\alpha = 0.85$, which suggests good inter-rater reliability within the research team.

To answer **RQ2**, we followed roughly the same inductive coding procedure for videos on linked list insertion. In addition to the four videos specific to linked list insertion, 9 of the 18 generic videos included some form of insertion. The videos were then broken down into clips covering the three different types of insertion for a linked list: prepend, insert anywhere, and append. The authors inductively coded each type of insertion separately. We achieved acceptable internal reliability ($\alpha = 0.94$) for all insertion codes.

4 RESULTS

We provide our full code book online [1]. We include a shorter, pictorial version of our code book in Table 2. We describe five related, but different, themes that emerged from the codes.

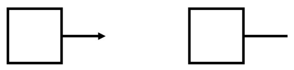
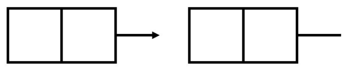
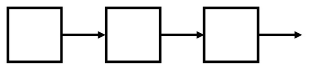
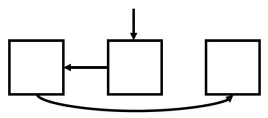
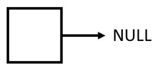
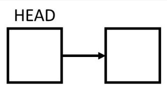
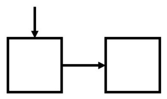
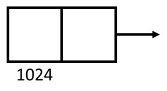
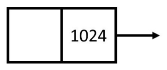
Code	Picture Example
Box and Arrow / Line	
Multi-box and Arrow / Line	
Left-to-right Linear	
Non-linear	
Arrow to NULL	
Head Label	
Head Arrow	
Address of Node	
Address of Next Node	

Table 2: Shorter, pictorial code book used to analyze diagrams

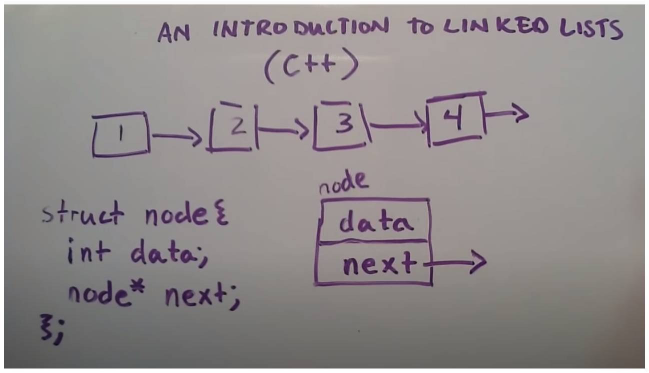
4.1 All YouTubers used a diagram

Every YouTuber used a diagram to visualize a linked list, though different YouTubers used diagrams for different purposes.

4.2 YouTubers used similar language with differing semantics

Almost all YouTubers used and defined canonical linked list terminology (e.g., "head", "tail," "node"), but there was little discussion on where these terms came from and little variation on the types of terms used. For example, Video #3 states, "A linked list is made up of nodes. Each node stores an item of data," but doesn't explain where the term "node" comes from. On the other hand, Video #13 acknowledges this ambiguity: "Each of these boxes we call nodes, that's just what they're called."

With "head" and "tail," YouTubers often alias these terms for the first and last node, respectively. For example, Video #4 states, "The head node points to the second node, which points to the



```
struct node{
  int data;
  node* next;
};
```

Figure 1: Cropped screenshot taken from Video #12. The YouTuber draws a box and arrow diagram (top), writes a code snippet (bottom left), then draws a multi-box and arrow diagram (bottom right).

third node, and so on until we reached the tail node that points to null, indicating the end of the list." However, this naming is misleading or confusing as "head" and "tail" are usually references or pointers, not the nodes themselves. When implementing linked lists, YouTubers had different layers of encapsulation, leading to subtle changes in the meaning of "head" and "tail." For example, Video #2 implemented a linked list object wrapper in Java: "We define a class, LinkedList, that's basically going to wrap our head." On the other hand, Video #10 implemented a node struct in C/C++: "The first node is also called the head node, and the only information that we keep all the time is the address of the head node or address of the first node." Both YouTubers used the word "head," but their differing implementations led to two different meanings.

Other common talking points among YouTubers included comparing linked list performance to arrays and their corresponding big-O analyses. Most YouTubers stored integers in their linked lists for simplicity, but mentioned linked lists can hold any data type. Some YouTubers mentioned different types of linked lists, such as doubly linked lists or sorted linked lists.

4.3 YouTubers move between diagrams of different and varying levels of abstraction

Some YouTubers opted to show different types of diagrams to highlight different layers of abstraction and detail. For example, Video #12 started with a box and arrow diagram, then wrote code for the node, then illustrated the code with a multi-box and arrow diagram (see Figure 1). Thus, the YouTuber started at a higher level of abstraction with less detail, then moved to a lower level of abstraction with more detail. On the other hand, Video #15 started with a detailed multi-box and arrow diagram while implementing the linked list, then illustrated linked list traversal with a simpler box and arrow diagram.

Next, we consider insertion videos. Analyzing insertion videos gave us clearer insight into how diagrams changed when applied to an algorithm.

For prepending, having a head reference becomes salient because that is the property of the list that is changing. From our analysis,

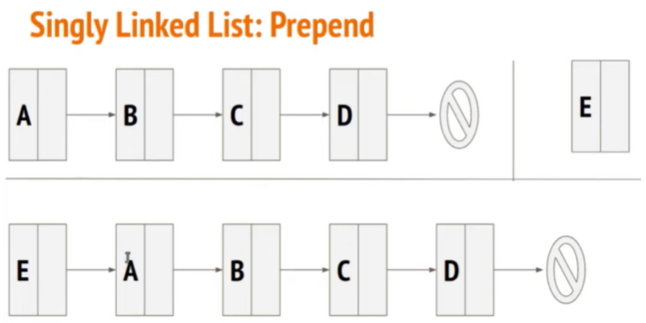
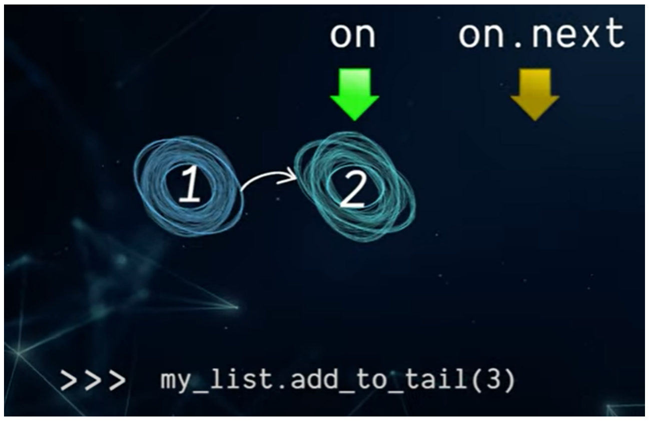


Figure 2: Cropped screenshot taken from Video #21. The YouTuber does not have a head reference while illustrating prepending.



```
>>> my_list.add_to_tail(3)
```

Figure 3: Cropped screenshot taken from Video #5. The YouTuber does not have an explicit null while illustrating appending.

4 of 7 clips consistently used a head reference, but 2 of 7 clips, both from the same video, added a head reference *during* the prepending process. Interestingly, 1 clip did not show a head reference at all while prepending and instead opted to show static before and after diagrams (see Figure 2).

For appending, having an explicit null (e.g., arrow to NULL) becomes salient to help identify which node is last or where the end of the list is, assuming a tail pointer is not used. All 5 clips analyzed did not use a tail pointer, and 3 clips consistently used an explicit null. However, 1 clip failed to have an explicit null at all while appending (see Figure 3), and 1 clip showed an explicit null prior to appending but failed to add it back after appending.

Most implementations of linked list in C/C++ incorporated addresses of nodes and next nodes into their diagrams but no implementations in Python or Javascript incorporated addresses. For examples in C/C++, having explicit addresses or a memory diagram may be more important to showcase the language's memory model (see Figure 4), but this lower level of abstraction may not be applicable to higher-level languages like Python or Javascript.

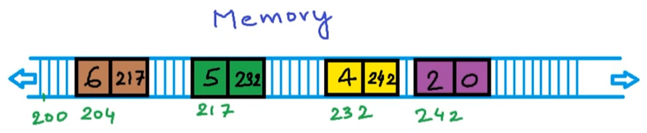
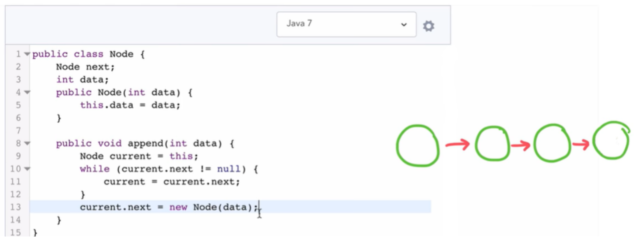


Figure 4: Cropped screenshot taken from Video #10. The YouTuber shows how a linked list fits in a 1D memory layout.



```
public class Node {
    Node next;
    int data;
    public Node(int data) {
        this.data = data;
    }

    public void append(int data) {
        Node current = this;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new Node(data);
    }
}
```

Figure 5: Cropped screenshot taken from Video #2. The YouTuber writes code that only manipulates one node at a time, but shows a diagram giving the illusion of access to all nodes at once.

4.4 Diagrams are subject to unexplained semantics that may not align with code semantics

Often, YouTubers assume diagram semantics to be self-evident, and they only label the diagram's parts rather than explain the rules of manipulation. Whereas code has a compiler to check for syntax and test cases to check for correct behavior, hand-drawn diagrams or animated slides do not have "verifiers" that enforce semantics and behavior. For example, when designing algorithms for linked lists, diagrams often give the illusion of having access to all nodes in the list at a time, whereas code typically has access to a single node at a time (see Figure 5). This easily leads to out-of-order assignment for an algorithm like insertion (see Figure 6). Additionally, many YouTubers used arrows to represent either a reference or an area for focus but did not explicitly specify which was which, aligning with Heiser and Tversky's [23] findings.

Insertion anywhere is arguably more complicated than prepending or appending to a linked list, as 6 out of 10 diagram-based clips analyzed showcased an *incorrect* algorithm, meaning if the diagram changes were translated pedantically to code, the resulting code would exhibit incorrect behavior. This seems to align with findings reported by Chotzen, et. al [8]. For example, Video #14 starts by creating a new node. However, the YouTuber reassigns the previous node's next pointer first (see Figure 6). In code, this would make the programmer lose access to the rest of the list, and thus not be able to assign the new node's next pointer correctly in the last step. This finding stands out especially when all 4 of the clips that showed code were implemented *correctly*, and only 2 out of those 4 warned about the perils of an out-of-order assignment.

Additionally, having a diagram leads to "diagram language." Treating the arrows or lines connecting nodes as "links" and therefore physical objects leads YouTubers to using inaccurate language

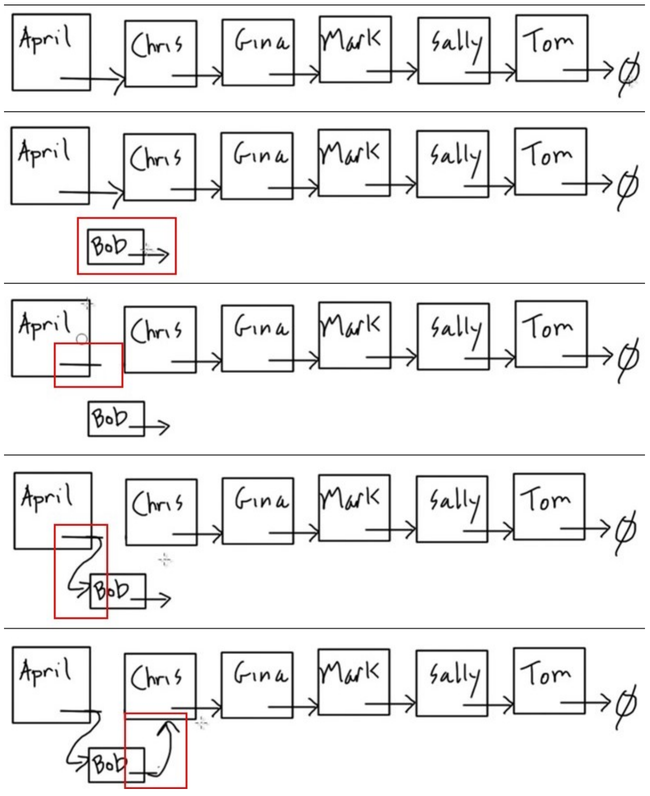


Figure 6: Cropped screenshots taken from Video #14. The YouTuber shows an incorrect insertion algorithm via out-of-order assignment. Annotations added in red.

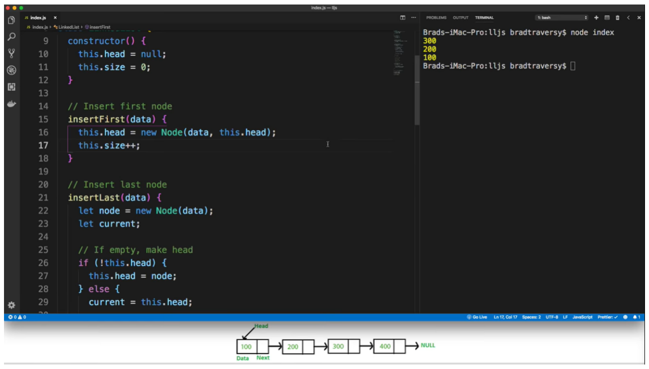
(e.g., "breaking," "removing," or "deleting" references) that has no direct mapping to code.

Another limitation of drawing algorithms in real time, whether on pen and paper, whiteboard, or digital canvas, is showing variable reassignment in two steps, when in terms of code, reassignment is viewed as a single step (see Figure 6).

4.5 YouTubers used diagrams according to different strategies

YouTubers used diagrams for different strategies when implementing algorithms in code. For example, Video #7 used a static image at the bottom of the screen while coding and referred to the diagram with a lot of virtual pointing via the mouse (see Figure 7). Video #13 introduced the topic with diagrams, then switched entirely to code. Video #5 showed the diagram, then the code, then an animated diagram alongside the code.

In our code book, we extrapolated these behaviors to 3 levels of interaction between code and diagram: no correspondence, ad-hoc correspondence, and one-to-one correspondence. For no correspondence, some videos were heavily introductory and had no code implementation. Others were focused on implementation and only showed a static diagram (see Figure 7). For ad-hoc correspondence, some YouTubers bounced between implementing the linked list and showing an example using the diagram, but the two were not



```
constructor() {
  this.head = null;
  this.size = 0;
}

// Insert first node
insertFirst(data) {
  this.head = new Node(data, this.head);
  this.size++;
}

// Insert last node
insertLast(data) {
  let node = new Node(data);
  let current;

  // If empty, make head
  if (!this.head) {
    this.head = node;
  } else {
    current = this.head;
```

```
Brads-iMac-Pro:lljs bradtraversy$ node index
300
200
100
Brads-iMac-Pro:lljs bradtraversy$
```

index.js — lljs
index.js
index.js › LinkedList › insertFirst
PROBLEMS OUTPUT TERMINAL
1: bash
Go Live Ln 17, Col 17 Spaces: 2 UTF-8 LF JavaScript Prettier: ✓
Head
100
200
300
400
NULL
Data
Next

Figure 7: Screenshot taken from Video #7. The YouTuber continuously references a static diagram at the bottom of the screen while implementing the linked list.

tightly coupled. For one-to-one correspondence, YouTubers stepped through their code and illustrated the corresponding change in the diagram, thus keeping the program state and diagram state in sync. The varying uses of diagrams in relation to code may indicate a preference for spatial thinking. If a YouTuber has no correspondence and little diagram interaction, then they implicitly require the viewer to mentally animate changes. If a YouTuber shows one-to-one correspondence, then there may be less reliance on a viewer's spatial ability.

5 DISCUSSION

5.1 What (if anything) should we standardize?

We might be tempted to think that diagram standardization is the answer to all of the variety, that one diagram will have all the features to be clear to any audience. This is certainly the case for molecular representations in chemistry: how one professor draws Fischer projections in one university is how another will draw the same Fischer projection. However, computer science is unique in this sense: we have a variety of programming paradigms and use cases for linked lists that may influence the types of diagrams we draw. We unpack some examples of these affordances below.

Different levels of abstraction may be better for different programming paradigms. For example, having a memory layout may make the C/C++ memory model more salient and easier to understand, but would be too low level for languages like Python or Javascript. Similarly, having explicit null values may be more applicable to C/C++ because memory management does not auto-initialize values, but less applicable in auto-initializing languages like Python and Java.

Different features become salient when implementing a linked list versus exploring theoretical points. For example, explicitly drawing the head, tail, and appropriate object wrappers may be more relevant when considering linked list implementation. On the other hand, a simpler diagram may be more useful and flexible when considering big-O analysis or comparing to an array.

Thus, moving toward a standardized linked list diagram for all data structures contexts may not be beneficial nor feasible. Instead, we may look towards standardizing diagram *semantics*. Many of the "mistakes" we found (i.e., incorrect algorithm when performing insertion on a diagram) were likely due to loose, unexplained, and unenforced diagram semantics. Agreeing upon a set of semantics may be easier for instructors and more helpful for students. However, defining the most beneficial diagrams and their associated semantics for a particular context or learning objective remains an open area for future work.

5.2 Uncommon Parameters

Uncommon parameters represent outliers in our analysis, but they question the "status quo" of linked list diagrams. For example, few videos drew a node or list wrapper to indicate encapsulation. Few videos drew non-linear linked lists to highlight the randomness of node location. Only one video showed a reference to a new node whereas many showed a reference to the first node. Few videos explained the perils of out-of-order assignment when inserting. However, we should not immediately take these outliers as innovations and incorporate them into our practice. Rather, we should carefully consider what the learning objectives are for our specific context and design diagrams accordingly.

6 LIMITATIONS

We acknowledge YouTube™ may not be a primary resource for traditional students studying in a university setting and may not be up-to-standard to some. However, given the prevalence and popularity of these videos, we believe there is a demand for these types of instructional videos that many find helpful. Additionally, we wanted to prevent personalized search results from potentially biasing data collection. While we cannot fully eliminate search bias, we believe that the "base" search bias will show what YouTube™ believes to be most relevant to our search terms.

7 CONCLUSION

Our analysis introduces a parameter space for instructors to continue thinking critically about how diagrams can enhance student learning and which features of a diagram are most salient to a particular learning objective. YouTubers and instructors alike seem to have intuitions about when certain abstractions or diagrammatic features are important for reasoning about linked lists. For example, different features may be helpful in certain contexts (i.e., head reference when prepending), but less relevant in others (i.e., head reference when appending). However, these intuitions are often not made explicit. Without explicit instruction on diagram semantics or the rules of manipulations, students may have to rely more on their spatial ability to mentally animate diagram transformations like in Figure 2 or instructors and students alike may make easy mistakes like out-of-order manipulations in Figure 6. Thus, we still need further research in understanding the affordances of different diagrams and what kinds of instruction using diagrams can especially support low spatial ability students.

ACKNOWLEDGMENTS

Acknowledgements removed for dual-anonymous review process.

REFERENCES

[1] [n.d.]. Code book (Public). https://docs.google.com/document/d/15_kqvyws9KGXjHhqHafRHbyIpCDR3KtdSAi0xx2XFXc/edit?usp=sharing

[2] ACM Computing Curricula Task Force (Ed.). 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM, Inc. https://doi.org/10.1145/2534860

[3] Derek Banas. 2013. *Linked List in Java.* https://www.youtube.com/watch?v=195KUinjBpU

[4] beiatrix. 2019. *Linked Lists | Data Structures in JavaScript.* https://www.youtube.com/watch?v=ChWWEncl76Y

[5] Ryan Bockmon, Stephen Cooper, Jonathan Gratch, Jian Zhang, and Mohsen Dorodchi. 2020. Can Students' Spatial Skills Predict Their Programming Abilities?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* ACM, Trondheim Norway, 446–451. https://doi.org/10.1145/3341525.3387380

[6] Ryan Bockmon, Stephen Cooper, William Koperski, Jonathan Gratch, Sheryl Sorby, and Mohsen Dorodchi. 2020. A CS1 Spatial Skills Intervention and the Impact on Introductory Programming Abilities. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20).* Association for Computing Machinery, New York, NY, USA, 766–772. https://doi.org/10.1145/3328778.3366829

[7] Sarah Buchanan and Joseph J. Laviola. [n.d.]. CSTutor: A Sketch-Based Tool for Visualizing Data Structures. 14, 1 ([n. d.]), 1–28. https://doi.org/10.1145/2535909

[8] Harrison Chotzen, Alasdair J. Johnson, and Parth M. Desai. 2019. Exploring the Mental Models of Undergraduate Programmers in the Context of Linked Lists. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19).* Association for Computing Machinery, New York, NY, USA, 1261. https://doi.org/10.1145/3287324.3293862

[9] CodeWhoop. 2016. *Linked List - Basics using C++.* https://www.youtube.com/watch?v=Zgzoe8jjidk

[10] Computerphile. 2017. *Linked Lists - Computerphile.* https://www.youtube.com/watch?v=_jQhALI4ujg

[11] Stephen Cooper, Karen Wang, Maya Israni, and Sheryl Sorby. 2015. Spatial Skills Training in Introductory Computing. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15).* Association for Computing Machinery, New York, NY, USA, 13–20. https://doi.org/10.1145/2787622.2787728

[12] CrashCourse. 2017. *Data Structures: Crash Course Computer Science 14.* https://youtu.be/DuDz6B4cqVc

[13] Vivekanand Khyade Algorithm Every Day. 2017. *Insert a node in Singly Linked List(at the start , middle or end).* https://www.youtube.com/watch?v=0xoYNbVTiSE

[14] Vivekanand Khyade Algorithm Every Day. 2017. *Introduction to Linked List in Data Structures (very easy).* https://www.youtube.com/watch?v=Rs1KPyb9fHY

[15] Sunil Dhimal. 2016. *Introduction to Linked List.* https://www.youtube.com/watch?v=zR6iIQnooP0

[16] CS Dojo. 2018. *Introduction to Linked Lists (Data Structures Algorithms 5).* https://www.youtube.com/watch?v=WwfhLC16bis

[17] Brian Faure. 2017. *Python Data Structures 2: Linked List.* https://www.youtube.com/watch?v=JlMyYuY1aXU

[18] Philip J. Guo. [n.d.]. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13* (Denver, Colorado, USA, 2013). ACM Press, 579. https://doi.org/10.1145/2445196.2445368

[19] HackerRank. 2016. *Data Structures: Linked Lists.* https://www.youtube.com/watch?v=njTh_OwMljA

[20] M. Hegarty. 2014. *Multimedia learning and the development of mental models.* Cambridge University Press, Cambridge, 673–702.

[21] M. Hegarty, M. S. Canham, and S. I. Fabrikant. 2010. Thinking about the weather: How display salience and knowledge affect performance in a graphic inference task. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 36, 1 (2010), 37–53. https://doi.org/10.1037/a0017683

[22] Mary Hegarty and Maria Kozhevnikov. [n.d.]. Types of Visual-Spatial Representations and Mathematical Problem Solving. 91, 4 ([n. d.]), 684–689. https://doi.org/10.1037/0022-0663.91.4.684

[23] Julie Heiser and Barbara Tversky. 2006. Arrows in Comprehending and Producing Mechanical Diagrams. *Cognitive Science* 30, 3 (2006), 581–592. https://doi.org/10.1207/s15516709cog0000_70 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog0000$_7$0

[24] Aaron Jack. 2020. *Basic DATA STRUCTURES Explained - LINKED LISTS.* https://www.youtube.com/watch?v=ne1iyAP9__o

[25] N. Johnson-Glauch and G. L. Herman. 2020. How engineering students use domain knowledge when problem solving using different visual representations. *Journal of Engineering Education* 109, 3 (July 2020). https://doi.org/10.1002/jee.20348

[26] Sue Jones and Gary Burnett. [n.d.]. Spatial Ability and Learning to Program. 4, 1 ([n. d.]), 47–61. https://doi.org/10.17011/ht/urn.200804151352

[27] K. Krippendorff. 2011. *Computing Krippendorff 's Alpha-Reliability.* https://repository.upenn.edu/asc_papers/43/

[28] LucidProgramming. 2018. *Data Structures in Python: Singly Linked Lists – Insertion.* https://www.youtube.com/watch?v=FSsriWQ0qYE

[29] Yukiko Maeda and So Yoon Yoon. 2013. A Meta-Analysis on Gender Differences in Mental Rotation Ability Measured by the Purdue Spatial Visualization Tests: Visualization of Rotations (PSVT:R). *Educational Psychology Review* 25, 1 (2013), 69–94. https://doi.org/10.1007/s10648-012-9215-x arXiv:https://doi.org/10.1007/s10648-012-9215-x

[30] Lauren E. Margulieux. 2019. Spatial Encoding Strategy Theory: The Relationship between Spatial Skill and STEM Achievement. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19).* Association for Computing Machinery, New York, NY, USA, 81–90. https://doi.org/10.1145/3291279.3339414

[31] Syeda F. Mazumder, Celine Latulipe, and Manuel A. Pérez-Quiñones. 2020. Are Variable, Array and Object Diagrams in Java Textbooks Explanative?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* ACM, Trondheim Norway, 425–431. https://doi.org/10.1145/3341525.3387368

[32] Traversy Media. 2019. *Linked List Data Structure | JavaScript.* https://www.youtube.com/watch?v=ZBdE8DElQQU

[33] mycodeschool. 2013. *Introduction to linked list.* https://www.youtube.com/watch?v=NobHlGUjV3g

[34] mycodeschool. 2013. *Linked List - Implementation in C/C++.* https://youtu.be/vcQIFT79_50

[35] mycodeschool. 2013. *Linked List in C/C++ - Inserting a node at beginning.* https://www.youtube.com/watch?v=cAZ8CyDY56s

[36] Miranda C. Parker, Amber Solomon, Brianna Pritchett, David A. Illingworth, Lauren E. Marguilieux, and Mark Guzdial. 2018. Socioeconomic Status and Computer Science Achievement: Spatial Ability as a Mediating Variable in a Novel Model of Understanding. In *2018 ACM Conference on International Computing Education Research.* ACM, Espoo, Finland, 97–105.

[37] Jack Parkinson and Quintin Cutts. 2018. Investigating the Relationship Between Spatial Skills and Computer Science. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18).* Association for Computing Machinery, New York, NY, USA, 106–114. https://doi.org/10.1145/3230977.3230990

[38] Jack Parkinson and Quintin Cutts. 2020. The Effect of a Spatial Skills Training Course in Introductory Computing. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20).* Association for Computing Machinery, New York, NY, USA, 439–445. https://doi.org/10.1145/3341525.3387413

[39] Paul Programming. 2012. *How to Create a Linked List C++ Introduction to Linked Lists.* https://www.youtube.com/watch?v=o5wJkJJpKtM

[40] ReelLearning. 2012. *Data Structures: Introduction to Linked Lists.* https://youtu.be/pBrz9HmjFOs

[41] Jacob Sorber. 2019. *Understanding and implementing a Linked List in C and Java.* https://www.youtube.com/watch?v=VOpjAHCee7c

[42] Sheryl Sorby, Beth Casey, Norma Veurink, and Alana Dulaney. 2013. The role of spatial training in improving spatial and calculus performance in engineering students. *Learning and Individual Differences* 26 (2013), 20 – 29. https://doi.org/10.1016/j.lindif.2013.03.010

[43] Sheryl A. Sorby. 2009. Educational Research in Developing 3â€D Spatial Skills for Engineering Students. *International Journal of Science Education* 31 (Feb. 2009), 459–480. Issue 3. https://doi.org/10.1080/09500690802595839

[44] M. Stieff, B. L. Dixon, M. Ryu, B. C. Kumi, and M. Hegarty. 2014. Strategy training eliminates sex differences in spatial problem solving in a stem domain. *Journal of Educational Psychology* 106, 2 (2014), 390–402.

[45] Mike Stieff and Sonali Raje. 2010. Expert Algorithmic and Imagistic Problem Solving Strategies in Advanced Chemistry. *Spatial Cognition & Computation* 10, 1 (2010), 53–81. https://doi.org/10.1080/13875860903453332 arXiv:https://doi.org/10.1080/13875860903453332

[46] Telusko. 2017. *5 Linked List Implementation in Java Part 1 | Data Structures.* https://www.youtube.com/watch?v=SMIq13-FZSE

[47] Telusko. 2017. *6 Linked List Implementation in Java Part 2 | Data Structures.* https://youtu.be/AeqXFjCUcQM

[48] David H. Uttal and Cheryl A. Cohen. 2012. Spatial Thinking and STEM Education: When, Why, and How? *Psychology of Learning and Motivation* 57 (2012), 147–181. https://doi.org/10.1016/B978-0-12-394293-7.00004-2 arXiv:https://doi.org/10.1016/B978-0-12-394293-7.00004-2

[49] Jonathan Wai, David Lubinski, and Camilla P. Benbow. 2009. Spatial Ability for STEM Domains: Aligning Over 50 Years of Cumulative Psychological Knowledge Solidifies Its Importance. *Journal of Educational Psychology* 101, 4 (2009), 817–835. https://doi.org/10.1037/a0016127 arXiv:https://doi.org/10.1037/a0016127

[50] Margaret Wilson. [n.d.]. Six views of embodied cognition. 9, 4 ([n. d.]), 625–636. https://doi.org/10.3758/BF03196322