Teaching and learning data structure concepts via Visual Kinesthetic Pseudocode with the aid of a constructively aligned app

Ogen Odisho, Mark Aziz, Nasser Giacaman
Department of Electrical and Computer Engineering
The University of Auckland, New Zealand

***Abstract*—Data Structures is an integral topic for any Computer Science or Software Engineering degree, identified as a Core Tier-1 topic of the ACM/IEEE Computer Science Curricula. The underlying concepts are inherently abstract, making them especially difficult to understand for novice programmers. This paper proposes a cognitively challenging technique to help students understand the thought process that the learning outcomes of fundamental data structure units aim to achieve. The development of this thought process is using a technique we term Visual Kinesthetic Pseudocode, with the overarching goal of helping students code without coding, yet providing the necessary scaffold to guide them in implementing the data structures with real code. This was implemented in the form of INTERACTIVEDS, an app for students and teachers to guide the learning of fundamental data structure concepts. The evaluations demonstrate that students strongly credited INTERACTIVEDS in aiding their understanding of concepts and confidence in applying data structure concepts in practice. The study is also a step forward in revealing potential threshold concepts pertaining to data structure modules.**

I. INTRODUCTION

The fundamental *data structures* unit is one of the most cognitively difficult topics in programming due to its high level of abstraction [1]. The learning is further exasperated since it builds on *object-oriented programming* principles and *pointers* – both of which are concepts identified as troublesome threshold concepts in computer science [2], [3]. While visualization has been attempted in helping students, they are deemed ineffective unless they cognitively engage and challenge students [4], [5]. This paper proposes *Visual Kinesthetic Pseudocode* (VKP) as an engaging technique to scaffold students transitioning "from concept to code".

Programming in general is a difficult subject to learn: it involves many abstract concepts and students rarely receive sufficient amounts of personal instruction [1]. The ACM and IEEE Computer Science Curricula [6] has identified data structures as one of the most essential topics, with the recommendation that 12 of the introductory hours dedicated to it. The issue in learning data structures lies in the high-level concepts rather than at the low-level programming technicalities [7]. Such topics are not limited to Computer Science and Software Engineering; Electrical and Electronics, Mechatronics, and Computer Systems Engineering are also examples incorporating CS2 level topics.

Since some of the difficulties associated with learning data structures is also acknowledged by teachers [1], [3], this raises some important questions. How can the underlying concepts of the fundamental data structure unit be taught, and therefore learnt by students, effectively? How can teachers ensure that their explanations are correct without inadvertent errors that could potentially further confuse students? Answering these questions requires a careful approach that is educationally sound, both in terms of motivating students as well as meeting the correct learning outcomes. Instructors teaching programming courses seek and welcome scaffolding tools that support their delivery of such abstract concepts [8]. This scaffolding also plays an essential role for students, particularly when attempting to implement the concepts with practical programming exercises.

A driving motivator for this work is that textual pseudocode lacks engagement and is easily misinterpreted [9]. VKP aims to reinforce the correct thought processes, which is a precursor for students understanding how to implement something in code. Using the ideas of VKP, INTERACTIVEDS has been implemented without losing sight on delivering the intended learning outcomes pertaining to data structures. INTERACTIVEDS ensures only programmatically-correct steps may be executed, with interactions strongly aligned to pseudocode such that any given action correlates to a "real line of code". This serves two purposes:

- Eliminate any inadvertent errors likely to arise when explaining data structure concepts, especially when the explanation is attempted in a visual manner.
- Provide a scaffolded learning environment to reduce ambiguity when transitioning from the concept to implementation, especially when conceptual explanations tend to be too abstract in helping students independently apply the concept in practice.

The rest of this chapter is organized as follows. Educational foundations guiding this work are overviewed in section II. Visual Kinesthetic Pseudocode is presented in section III, before introducing INTERACTIVEDS in section IV. Student experiences and potential threshold concepts are discussed in section V before presenting a brief overview of related work in section VI, then concluding in section VII.

II. BACKGROUND

A. ACM & IEEE CS2013: Fundamental Data Structures unit

The ACM and IEEE Computer Science Curricula (CS2013) [6] is a guideline for Computer Science and Software Engineering undergraduate degree curriculum design. The core topics are decomposed into Core Tier-1 and Core-Tier 2, with most of the Core Tier-1 topics covered in introductory courses. CS2013 identified 165 hours dedicated to Core Tier-1 topics, 43 of which are dedicated to Software Development

Fundamentals (SDF). Of these 43 hours, 12 are dedicated to the Fundamentals Data Structures unit. Among others, some topics in this unit include: (i) Arrays, (ii) Abstract data types and their implementation, (iii) Linked lists. The unit defines Learning Outcomes, a couple of selected examples include:

LO#3 Write programs that use each of the following data structures: arrays, records/structs, strings, linked lists, stacks, queues, sets, and maps. *[Usage]*

LO#4 Compare alternative implementations of data structures with respect to performance. *[Assessment]*

Each learning outcome is annotated with level of mastery using a somewhat simplified Bloom's taxonomy [10]. In increasing order of mastery, they are *Familiarity* (awareness of concept), *Usage* (apply concept in a practical manner), and *Assessment* (understand alternative options and justify their selection). This presents a challenge for novice programmers: the expected level of mastery of the learning outcomes is notably high. As a Core Tier-1 unit, many students may still be trying to familiarize themselves with the programming language technicalities. But before students can code these data structures to demonstrate usage mastery, an obvious prerequisite is a strong understanding in the underlying concept.

B. INTERACTIVEDS *educational theories*

1) Levels of thinking about teaching: The primary design decision for INTERACTIVEDS was to promote an *active learning* approach. This is motivated by the three levels of thinking about teaching that an instructor may take in teaching [11]:

1) What the *student is*: the teacher believes responsibility to learn lies solely with students, and little can be done to motivate them.
2) What the *teacher does*: the teacher attempts to interest students by focusing on the way material is presented.
3) What the *student does*: the teacher recognizes that learning comes from student engagement.

To foster learning, focus needs to be on *what the student does* [12]. Here, students learn by actively applying older knowledge as they encounter new knowledge [13]. Such active learning is vital for long-term retention, in which students make meaning of the new information by relating it to existing knowledge [14]. INTERACTIVEDS promotes this, as it comes in the form of activities to engage students, but is still also useful as a teaching aid for the instructor when explaining concepts in class.

2) Constructive Alignment: Engagement is not the only ingredient required for learning; an engaged student may not necessarily be learning the intended outcomes. *Constructive alignment* [15]. Rather than focusing on topics the teacher should teach, constructive alignment first focuses on the *Intended Learning Outcomes* (ILO): what students should learn and to what extent. In the case of the data structures topic, the learning outcomes are provided by the CS2013 learning outcomes (as discussed in section II-A). The second step of constructive alignment involves the identification of *Activities* that engage students in order to meet those ILO. LO#3 states that students should be able to "*write programs that use the data structures*". While an obvious activity might entail a lab exercise requiring students to code and use the data structure, this is likely to be a daunting task for weaker students. INTERACTIVEDS is built on activities that serve as stepping stones. Finally, *Assessment Tasks* are required to provide feedback informing students about their learning progress. In the context of INTERACTIVEDS, this is implemented with randomly generated exercises to manipulate the data structures. This process of accomplishing sub-goals (that lead to larger goals) helps promote student self-motivation, ultimately nurturing self-efficacy [16], [17].

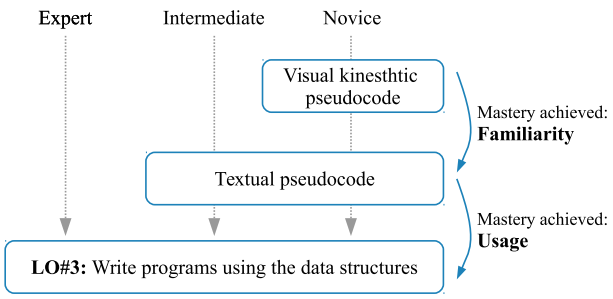


Figure 1. Support steps to achieve CS2013 Learning Outcome #3. VKP provides a stepping stone for novice programmers to achieve *Familiarity*, which is a prerequisite before students achieve the *Usage* mastery level.

III. VISUAL KINESTHETIC PSEUDOCODE

This section introduces *Visual Kinesthetic Pseudocode* (VKP) as a strategy to scaffold novice programming students when studying abstract and concept-rich topics that require a high level of mastery. The goal is to provide a stepping stone in understanding the concept (i.e. the thought process), as this is a prerequisite to the actual coding (i.e. practical application of the concepts). This is illustrated in Figure 1, showing the support steps a student requires before being able to apply to real programming exercises (such as LO#3). Although the intention of textual pseudocode is to make it easier to understand real code by expressing it informally, it still possesses some problems. Novice programming students find pseudocode easy to misinterpret, and it does little in helping them detect their misunderstanding due to the lack of feedback it provides [9].

Applying VKP is not limited to learning outcomes that incorporate code development; VKP can also provide a stepping stone when targeting a high level of conceptual understanding, such as appreciating the performance consequences of alternative data structure implementations (e.g. LO#4). The kinesthetic aspect of VKP would allow students to "experience" the performance associated with different implementations. The important components of the VKP strategy are:

- **Visual:** This recognizes that code (whether it be real code or pseudocode) can be daunting for novice programmers, and the peculiarities surrounding each programming language's syntax only further inhibits understanding of the underlying concepts. While textual pseudocode has the benefit of being programming language-neutral, visualizations help students develop mental models [5].
- **Kinesthetic:** Visualizations alone are insufficient in helping students learn; for them to be effective, they need to cognitively engage and challenge students [4], [5]. This component therefore promotes active learning.

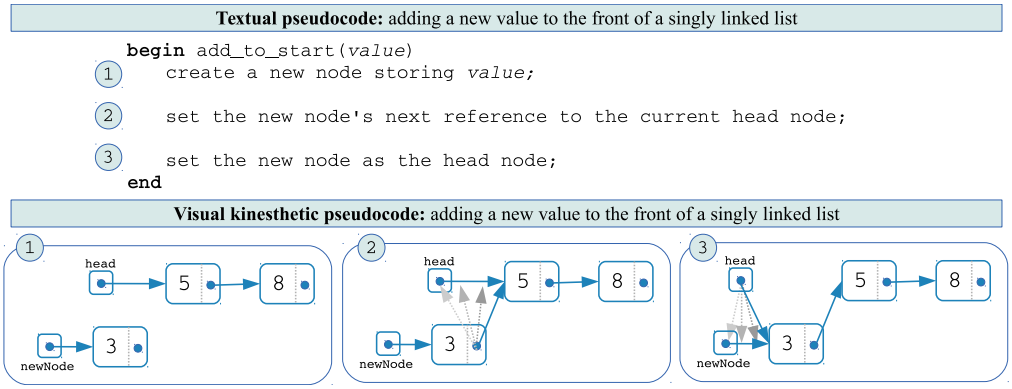


Figure 2. An example incorporating visual and kinesthetic considerations to produce a VKP. The careful alignment to the textual pseudocode ensures the interactive visualization meets the intended learning outcome. While the VKP is more engaging and less daunting for novice programmers, the direct alignment will still allow the automatic generation of textual pseudocode from the corresponding actions.

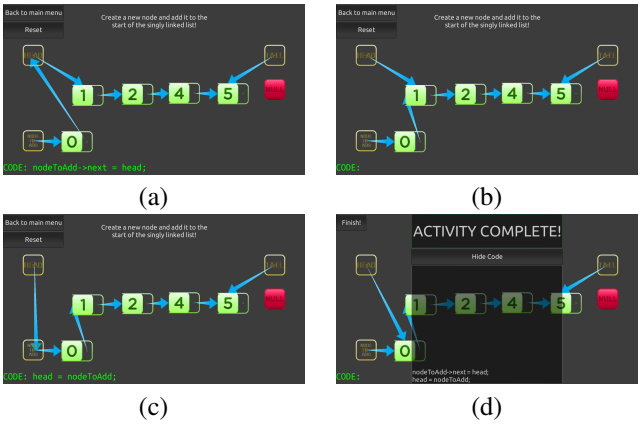


Figure 3. An activity using VKP to add a new node to the front of a singly linked list. Each kinesthetic action is directly related to corresponding code (shown at the bottom). The "sliding" effect (e.g. Step 2 in Figure 2) is shown in (a) to (b). This aids the understanding of pointers, by avoiding the incorrect approach of attempting to directly access nodes.

- **Pseudocode:** For engagement to contribute to learning, it needs to be closely aligned to the ILO. The Fundamental Data Structures unit requires a high level of mastery that requires students to apply the thought process in manipulating data structures. This process, if documented as pseudocode, is a helpful precursor to writing "the real code" [18]. The difference with VKP is that pseudocode is constructed by students visually *and* kinesthetically.

Aligning visualization and kinesthetics to pseudocode: Figure 2 illustrates traditional textual pseudocode for a learning activity to manipulate a data structure, in this case adding a new value to the front of a singly linked list. This example illustrates the potential misinterpretation that can result with using textual pseudocode; does line 3 mean "`head = newNode`", or does it mean "`newNode = head`"? Due to the informal language of pseudocode, it is easy to see how misinterpretation might creep in. Below the textual pseudocode is the same learning activity, this time infused with visual and kinesthetic components to produce the VKP. Here, misinterpretation has been eliminated since each kinesthetic action is intentionally designed to be mapped to real code.

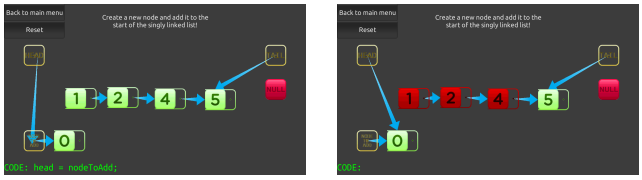


Figure 4. The learner's *goal* is to add a new node to the front of the singly linked list. The learner's first step in the *plan*, making `nodeToAdd` as the `head`, immediately results in an *expectation failure*.

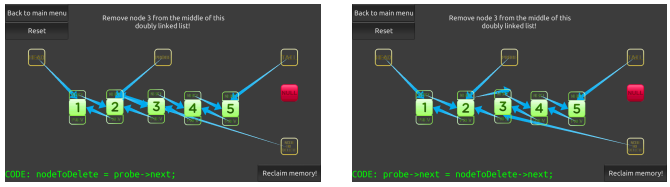
IV. INTERACTIVEDS

The design recommendations discussed in the previous sections have been implemented in INTERACTIVEDS[1], in the form of an app freely available for both students and teachers. Figure 3 illustrates INTERACTIVEDS screenshots in correctly adding to the front of a singly linked list. Each step in the VKP interaction (outlined in Figure 2) directly maps to pseudocode.

Learning with expectation failure: To promote deep learning, students need to have their existing mental models challenged [19]. Long term understanding is aided with the intellectual stimulation of grappling with a problem. As an essential part of this learning process, INTERACTIVEDS capitalizes on the concept of "expectation failure" [20]. This is where the learner has a *goal*, follows a *plan*, but fails when the result does not meet their *expectation*. This is best illustrated with Figure 4, where the user's goal is to add a node to the front of a singly linked list. It may seem simple enough, but this particular activity catches out many users when attempting it for the first time. When the wrong action is taken, the linked list loses reference to the first 3 elements. INTERACTIVEDS recognizes that the data structure is in an incorrect state, so it immediately highlights the lost nodes and vibrates to inform the user (Figure 4(b)).

Reducing misconceptions: A challenge in learning a conceptual topic is that, if the explanation is too abstract (in the aim of focusing on the conceptual explanation), it does little in helping the learner *apply* the concepts. Furthermore, visually explaining concepts may lead to misconceptions when the visualizations do not correctly reflect the intended and correct

[1] www.ece.auckland.ac.nz/~ngia003/dsapp



(a) Traversing to middle (b) Updating nodes

Figure 5. Deleting the middle of a doubly linked list involves navigating to the respective node using a `probe` pointer, ensuring that users cannot directly access it even though it is visually "there".

meaning. For example, when removing a node from the middle of a linked list, it may be tempting to "directly point" to the middle node. However, just because "we can see it" in the visualization, it does not mean we can actually access it yet. The kinesthetic interactions of INTERACTIVEDS help learners understand they cannot manipulate or directly reference nodes until they access them by traversing the data structure; Figure 5 shows two steps of this activity, here the use of a `probe` pointer to navigate to the middle of the doubly linked list. If a teacher was explaining this activity without reinforcing these rules (e.g. by using a freehand sketch), students might be misled into believing operations on the middle of a linked list are constant-time operations.

Another added challenge in learning how to manipulate data structures is its heavy use of pointers, which is a threshold concept in itself that novice programmers struggle with [2], [3]. Consider again Figures 3(a) and 3(b). In setting the value of `nodeToAdd->next`, the user learns that they cannot directly point to Node 1; instead, the value of `head` is used (Figure 3(a)), which effectively causes it to point to the node (hence the "sliding" effect that results in Figure 3(b)). INTERACTIVEDS ensures that the correct intention of using pointers is respected, avoiding misunderstanding of pointers.

V. EVALUATION

INTERACTIVEDS was offered to a CS2 course consisting of 250 students. The course is compulsory and includes Electrical and Electronic, Mechatronics, and Computer Systems Engineering students. The data structures unit spanned two teaching weeks, focusing on the core learning outcomes presented in section II-A. The logs in this section were gathered from the Android and WebPlayer platforms, from 159 and 38 unique devices respectively. At the end of the data structures unit, students were asked to complete an anonymous questionnaire delivered in-app, where 49 responses were received.

Limitations: It was an informed design decision to encourage student uptake by ensuring privacy was respected. Since data gathered was anonymous, this meant that the questionnaire and activity logs could not be correlated to particular individuals, let alone to course assessments. Although individuals could not be identified, it was possible to distinguish usages on different devices (based on unique device ID). It is therefore possible that a student may have installed the app on multiple devices (e.g. a tablet *and* a smartphone), or used the WebPlayer from multiple browsers; in such cases, this study treats different devices as different users. Only logs coinciding with the delivery of the data structure module are included, maximizing the likelihood that they were from students genuinely enrolled in the course. Finally, receiving logs was at the user's discretion: users are allowed to opt out of sending logs by disabling this from within the app settings. While the app *may* have been used more than is portrayed by the logs, but it is not possible to determine how many users disabled sending anonymous data.

A. Student perceptions

Table I summarizes anonymous responses (using the 5-point Likert scale) from the in-app questionnaire. The questionnaire was visible only to users that completed at least half the activities (to ensure only genuine users completed it). The app most effectively developed student confidence for Array, Vector and Linked List activities (Q1-Q3). These activities were the most cognitively challenging activities, requiring users to interact using the VKP technique proposed earlier. This is best illustrated with the following student comment, emphasizing learning through *expectation failure*:

> *"I found this easy to learn because I was able to interact with the data structure, make mistakes, then see how the order of doing things played a part."*

The other activities were less cognitively challenging, especially the Queue and Stack activities since the aim of their semi-automated interaction was to emphasize the code-reuse of other data structures (Q5). Many students commented on the lack of interactivity for these activities, therefore confirming mobile app activities need to be cognitively challenging:

> *"The fact they were only animations as opposed to exercises meant less thinking was required."*

Q6-Q10 show that students generally agreed that data structure concepts were effectively conveyed with INTERACTIVEDS. The CS2013 Learning Outcomes however require a much deeper level of understanding; in this regards, Q12 and Q13 show that INTERACTIVEDS helped provide some scaffolding for students to bridge concepts with coding activities. This is emphasized with Q14, where a fair portion of students actually wanted to see more code. The app was useful in guiding students towards implementing with code:

> *"Being visually able to work through linking forward and behind, then deleting the nodes, really helped concrete the process required to then implement it into my lab and assignment. Also being able to see pseudocode after completing the task is a real help."*

Some students hinted they wanted to see more engaging and challenging exercises (Q15), but otherwise had a highly positive attitude to the educational benefits of using mobile apps in their learning (Q16-Q19). The combination of visual *and* kinesthetic pseudocode was a powerful combination:

> *"The use of visual diagrams with code provided after successful completion showed what was going on with each step. Drag and drop feature made it clear what steps I was doing to get my intended result."*

B. Logged usage

During the two week data structure module, over 5600 activity logs were recorded from the Android and WebPlayer platforms. Table II shows the time distribution of all activity attempts, to help convey the granularity of engagement.

Question	SA	A	N	D	SD
Self-efficacy of particular topics					
Q1. The app gave me confidence understanding how Arrays and Vectors work	26	20	3	0	0
Q2. The app gave me confidence manipulating a Singly Linked List	30	16	2	1	0
Q3. The app gave me confidence manipulating a Doubly Linked List	21	21	6	1	0
Q4. The app gave me confidence understanding Circular Arrays	21	16	10	2	0
Q5. The app helped me understand how encapsulation helps us implement Queues and Stacks	9	27	11	2	0
Effectiveness in aiding the understanding of data structure concepts					
Q6. The app helped me understand data structure concepts	25	23	1	0	1
Q7. The visual aspects of the app helped me understand underlying concepts of data structures	29	17	2	0	1
Q8. The app helped me understand the underlying steps of data structure manipulation without being daunted by code	29	15	3	2	0
Q9. The app helped me to easily follow lecturer's teaching flow during the class	16	25	7	1	0
Q10. The activities in the app were aligned well with the Learning Outcomes of the data structures module	17	29	3	0	0
Scaffolding towards practical implementation					
Q11. The app helped me to study the data structures module of the course more systematically	12	25	10	2	0
Q12. The app helped me prepare for the data structure laboratory exercises	18	24	5	2	0
Q13. The app helped give me confidence in the underlying concepts so I can go implement them in code	24	23	2	0	0
Q14. I really liked that there was not much code shown in the app, with focus being on the data structures concepts	15	12	7	14	1
App interaction and overall impression					
Q15. I found the random exercise testing mode engaging and challenging	14	22	13	0	0
Q16. I enjoyed the drag & drop aspects of the app	23	22	3	0	1
Q17. I wish similar educational mobile apps would be used in other programming courses	25	21	2	1	0
Q18. I found that using a mobile phone app to study course material outside of class time was convenient	25	17	6	1	0
Q19. Should a friend be studying a similar data structures course, I would definitely recommend the app to them	28	21	0	0	0

Table I
SELECTED QUESTIONS FROM 49 STUDENTS IN THE ANONYMOUS INTERACTIVEDS QUESTIONNAIRE

Time band	Completed	Attempted	% Completed
0-2 seconds	0	619	0%
3-5 seconds	80	301	27%
6-10 seconds	217	550	39%
11-20 seconds	446	906	49%
21-30 seconds	334	766	44%
31-59 seconds	541	1298	42%
1-10 minutes	435	1097	40%
10+ minutes	15	76	20%
Total	2068	5613	37%

Table II
DISTRIBUTION OF TIME ON INTERACTIVEDS ACTIVITIES

	Front		Middle		End	
	+	-	+	-	+	-
Proportion of attempts successfully completed						
SLL	228/632	94/353	115/492	63/331	172/405	106/225
	36%	27%	23%	19%	42%	47%
DLL	55/136	55/147	38/85	36/129	56/115	39/67
	40%	37%	45%	28%	49%	58%

Table III
COMPLETION RATES FOR LINKED LIST OPERATIONS

For each given time band, the table shows the proportion of attempted activities that were successfully completed (as opposed to resetting the activity when a mistake is made, or the user exiting the activity without completing it). The app was purposely designed such that activities were digestible, in the aims of encouraging engagement by having simple activities that are quick to complete. In this trial, 78% of the activities successfully completed were accomplished in under a minute. It is assumed that completions exceeding 1 minute were probably students coding an assignment or lab exercise, and were using the app to guide their programming (as opposed to struggling to complete the activity). It is worth noting that there is no extrinsic incentive to complete activities (i.e. gamification); there is no visible scoring system informing users on the number of completed or abandoned activities.

Table III more specifically looks at the completion rates for the linked list activity attempts, but only includes attempts at least 5 seconds long (to filter out probable inadvertent attempts when a user immediately exits an activity). A total of 2438 attempts were made for Singly Linked List (SLL) operations, while only 679 attempts were made for Doubly Linked List (DLL) operations. For SSL, the completion rates suggest that operations in the middle of the SLL are most challenging, while operations at the end of the SLL are least challenging. It may come as a surprise that the successful completion rates of the SLL operations are higher than that of the DLL operations, but this is likely attributable to predominantly stronger students attempting DLL activities. Removing from the middle of a DLL is clearly the most challenging operation, while operations on the end of the DLL least challenging.

For a user having successfully completed a given linked list activity, we next investigate the total number of overall app activities completed. This aims to understand if the user's general engagement is an attribute that contributes to them completing the more challenging activities. Building on the assumption made earlier that DLL activities are predominantly targeted by stronger users, we distinguish between two groups of users: those that have successfully completed half of DLL activities (namely 3+ of the 6), versus those that have not. Figure 6 shows a significant distinction of general engagement in completing activities between the two groups. The value above each whisker is the total number of students in that group that successfully completed the respective linked list operation, while the y-axis shows the group's distribution of overall app engagement. This figure helps explain, for example, why the DLL completion rates are higher than the SLL rates for operating in the middle of the linked list; it is due to the DLL being predominantly targeted by the more engaged (and presumably stronger) users.

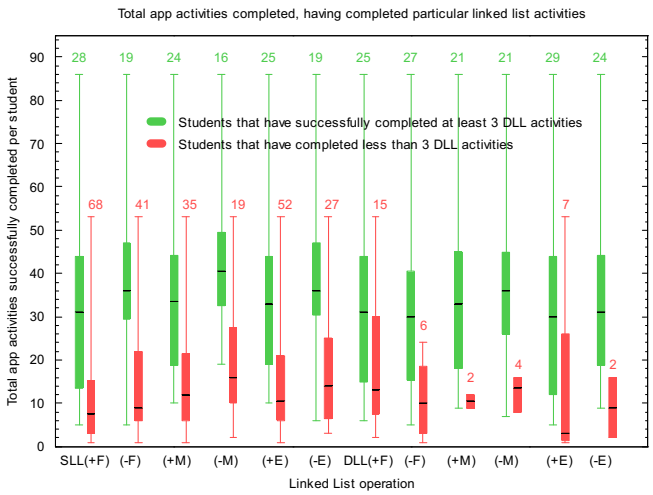


Figure 6. General engagement of completing INTERACTIVEDS activities, based on having completed a particular linked list activity. The users were grouped into two categories: those that managed to complete at least 3 DLL activities, versus those that did not.

VI. RELATED WORK

Helping students learn programming concepts with the help of visualization is not new [21], and is best known in popular visual programming tools such as Alice [22] and Scratch [23]. Early studies have investigated the effects of algorithm and program visualization and have concluded that cognitive engagement is the missing element desired to achieve learning [4], [5]. By focusing specifically on data structures, and acknowledging the wide variation of philosophy on teaching data structure concepts [24], this paper proposes VKP to aid learning. Despite these teaching variations, the underlying learning outcomes of CS2013 [6] need to be respected. A combination of competitive programming and game development incorporated into course assessments has been used in motivating students in a data structures course [25] by comparing the code they develop against the instructor and other students. Similarly, the JDSL Visualizer [26] is a testing tool for students to see the effect of their developed data structures. However, our focus in this paper is on the prerequisite step: the scaffolding students need in order to absorb data structure concepts since many struggle with the programming strategy.

VII. CONCLUSIONS

Learning a conceptual programming topic, such as the Fundamental Data Structures unit in the ACM & IEEE Computer Science Curricula, presents multiple challenges. First, students need to master the thought process underpinning the concepts. While visualizations may first appear as a promising aid, engagement with the visualization is essential in cognitively challenging the learner. The second challenge is the transition, from having firstly understood the concept, to developing a mastery level demonstrating an ability to apply the concepts in practice. Visual Kinesthetic Pseudocode has been proposed as a technique to address these concerns while also reducing potential confusion arising from inadvertently wrong explanations. The implementation, in the form of an app called INTERACTIVEDS, has demonstrated a promising approach in scaffolding novice programmers to learn an abstract topic such as data structures. The activity logs also identified the most challenging operations in manipulating linked lists.

REFERENCES

[1] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *SIGCSE Bull.*, vol. 37, pp. 14–18, June 2005.

[2] J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, M. Ratcliffe, K. Sanders, and C. Zander, "Threshold concepts in Computer Science: do they exist and are they useful?," *ACM SIGCSE Bulletin*, vol. 39, no. 1, pp. 504–508, 2007.

[3] I. Milne and G. Rowe, "Difficulties in learning and teaching programming–views of students and tutors," *Education and Information technologies*, vol. 7, no. 1, pp. 55–66, 2002.

[4] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[5] M. D. Byrne, R. Catrambone, and J. T. Stasko, "Evaluating animations as student aids in learning computer algorithms," *Computers & education*, vol. 33, no. 4, pp. 253–278, 1999.

[6] ACM and IEEE, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, 2013.

[7] M. Butler and M. Morgan, "Learning challenges faced by novice programming students studying high level and low feedback concepts," in *ascilite Singapore 2007 ICT: Providing Choices for Learners and Learning*, pp. 99–107, Nanyang Technological University, 2007.

[8] N. B. Dale, "Most difficult topics in CS1: results of an online survey of educators," *ACM SIGCSE Bulletin*, vol. 38, no. 2, pp. 49–53, 2006.

[9] T. Hübscher-Younger and N. H. Narayanan, "Constructive and collaborative learning of algorithms," in *ACM SIGCSE Bulletin*, vol. 35, pp. 6–10, ACM, 2003.

[10] L. W. Anderson and D. R. Krathwohl, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Pearson, 2000.

[11] J. Biggs and C. Tang, *Teaching for quality learning at university*. Open university press, 3rd ed., 2007.

[12] J. Biggs, "What the student does: teaching for enhanced learning," *Higher education research & development*, vol. 18, no. 1, pp. 57–75, 1999.

[13] R. E. Mayer, "The psychology of how novices learn computer programming," *ACM Comput. Surv.*, vol. 13, pp. 121–141, Mar. 1981.

[14] E. F. Barkley, *Student Engagement Techniques: A Handbook for College Faculty*. Jossey-Bass, 2009.

[15] J. Biggs, "Enhancing teaching through constructive alignment," *Higher Education*, vol. 32, no. 3, pp. 347–364, 1996.

[16] A. Bandura and D. H. Schunk, "Cultivating competence, self-efficacy, and intrinsic interest through proximal self-motivation," *Journal of personality and social psychology*, vol. 41, no. 3, pp. 586–598, 1981.

[17] B. J. Zimmerman, "Self-efficacy: An essential motive to learn," *Contemporary educational psychology*, vol. 25, no. 1, pp. 82–91, 2000.

[18] G. G. Roy, "Designing and explaining programs with a literate pseudocode," *Journal on Educational Resources in Computing (JERIC)*, vol. 6, no. 1, p. 1, 2006.

[19] K. Bain, *What the best college teachers do*. Harvard University Press, 2011.

[20] R. C. Schank, T. R. Berman, and K. A. Macpherson, "Learning by doing," *Instructional-design theories and models: A new paradigm of instructional theory*, vol. 2, pp. 161–181, 1999.

[21] M. Ben-Ari, R. Bednarik, R. B.-B. Levy, G. Ebel, A. Moreno, N. Myller, and E. Sutinen, "A decade of research and development on program animation: The Jeliot experience," *Journal of Visual Languages & Computing*, vol. 22, no. 5, pp. 375–384, 2011.

[22] W. P. Dann, S. Cooper, and R. Pausch, *Learning to Program with Alice*. Prentice Hall Press, 2011.

[23] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[24] R. Lister, I. Box, B. Morrison, J. Tenenberg, and D. S. Westbrook, "The dimensions of variation in the teaching of data structures," in *ACM SIGCSE Bulletin*, vol. 36, pp. 92–96, ACM, 2004.

[25] R. Lawrence, "Teaching data structures using competitive games," *Education, IEEE Transactions on*, vol. 47, no. 4, pp. 459–466, 2004.

[26] R. S. Baker, M. Boilen, M. T. Goodrich, R. Tamassia, and B. A. Stibel, "Testers and visualizers for teaching data structures," in *ACM SIGCSE Bulletin*, vol. 31, pp. 261–265, ACM, 1999.