What's Motivation Got to Do with It? A Survey of Recursion in the Computing Education Literature

Amber Settle
DePaul University
243 S. Wabash Avenue
Chicago, IL 60604
+1 (312) 362-5324
asettle@cdm.depaul.edu

ABSTRACT
One of the most challenging topics for both computing educators and students is recursion. Pedagogical approaches for teaching recursion have appeared in the computing education literature for over 30 years, and the topic has generated a significant body of work. Given its persistence, relatively little attention has been paid to student motivation. This article summarizes results on teaching and learning recursion explored by the computing education community, noting the relative lack of interest in motivation. It concludes by briefly discussing an approach to teaching recursion is appealing for students interested in web development.

1. INTRODUCTION
One of the most widely studied topics among computing educators is programming pedagogy. While the mix of topics studied changes, some subjects continue to elicit interest from researchers after many decades of work. Typically these problems are the ones that elude straightforward solutions, and a programming topic that has proven to be one of the most difficult to master is recursion [8]. Nearly every computing educator who writes about recursion notes that it is difficult to teach [9, 14], that it is difficult for students to learn [10, 12, 13, 15, 17, 18, 19, 25, 26, 29, 32, 36, 37, 38, 40] or both [16, 20, 27, 39], although one dissenter claims students only think recursion is difficult when instructors suggest it [2]. Regardless of whether it is the teaching or the learning that constitutes the main challenge, the combination of approaches for teaching recursion and the degree to which students master the topic has generated a significant body of work in the computing education community.

This interest in recursion is natural since recursion is a fundamental topic in the computing curriculum [6] and included in the information technology curriculum [21]. While it is a long-standing and prominent approach to solving problems, there is no single approach that appears to work for all audiences. Perhaps more interesting is the lack of attention paid to the motivational aspects of learning recursion. It has been shown that there is a relationship between student motivation and learning to program [4], and authors who consider broader programming pedagogy consistently discuss motivational aspects. It is therefore surprising that relatively little attention has been paid to student motivation and recursion. In this article we summarize results on teaching and learning recursion explored by the computing education community, discussing various approaches that have been taken for improving pedagogy and student learning. We note the relative lack of focus on student motivation, which suggests that motivational aspects of learning recursion may be understudied, and conclude by summarizing an approach to recursion that uses web development as a motivator.

2. LITERATURE REVIEW
Recursion is a well-studied topic in computing education, with articles dating to at least the 1980s. The focus of each researcher varies, with novel pedagogical approaches, fruitful and illuminating problems, students' mental models for recursion, the relationship between non-traditional populations and recursion, and the impact of recursion on interest in computing the main themes for the overall body of research. In this section we summarize the contributions to these areas.

2.1 Novel Pedagogical Approaches
Many authors consider novel pedagogical approaches to teaching recursion. CS Unplugged [7] activities are one venue for teaching recursion, both to traditional college-age students [14] and to 11-14 year-olds [17]. A notable paper suggests that problems lending themselves to dramatization with a clear link to the algorithm have promise for improving student understanding of recursion [1].

One line of research suggests that a focus on structural recursion is crucial [2, 3, 14] rather than the more common method-based recursion. As a part of a larger paper on approaches to teaching linked lists, Bloch argued that the most natural way to introduce recursion is using recursive data structures [2]. Bruce and his colleagues argue that structural recursion should be taught in CS1 courses prior to arrays, both as a way to better motivate the development of recursive approaches and as a way of reinforcing encapsulation during object-oriented design [3]. Other authors followed with class activities that developed a recursive list class in Python, building on students' knowledge of the built-in list class and employing active-learning techniques [14].

2.2 Effective Problems
Finding effective recursion problems is a focus of some authors. An early article considered the use of Prolog in combination with fractals, such as Koch's snowflake [10], and another suggested that the use of trees would enable students to decide when recursion could be effectively employed [24].

It has been suggested that combinatorial problems [31] or combinatorial counting equivalence problems [30] are particularly well-suited to recursive decomposition. One researcher hypothesizes that recursion graphs, modified trees that represent a sequence of recursive calls in a detailed and formal way, are

productive in helping students visualize recursion [20]. Another approach for helping students to visualize recursion is the use of recursively-generated geometric designs where the visual output of all the recursive calls can be seen [15]. The idea is to help convey state information to the students in as clear a way as possible. One study found that the animation of algorithms was helpful in engendering transfer for recursion problems but only when the approach was taken as a part of the overall learning environment [23].

The argument that the problem should lend itself naturally to recursion is made by an author who suggests that the problem of randomly parking cars is particularly effective in this regard [38]. This idea is also considered by an author who suggests that graphical problems for which iterative solutions are complex can be highly motivating, including Sierpinski's Triangle [37]. His argument is that visual problems with recursive solutions that are at least as simple as iterative ones provide students with early examples of the strength of recursive techniques.

In a connection to unusual pedagogical approaches, one author suggests that real-world problems with a strong connection to situations that can be acted out by students have the potential to improve understanding of recursion [1]. The problems mentioned include recognizing balanced parentheses, computing factorials, and searching an array.

2.3 Designing Pedagogical Approaches

Productive approaches to convey recursive thinking to students is the focus of numerous studies. Earlier work especially emphasizes the importance of taking a high-level approach to recursion, one that separates it as much as possible from the machine-level implementation [13], with one article arguing that showing the correctness of recursive algorithms could be done using abstraction and mathematical induction [11].

A suggestion that ML is the best language for teaching recursion was made by one pair of authors, who argued that the language lends itself naturally to experimentation, allows polymorphism, and provides mechanisms for defining recursive data structures [19].

One author found that an emphasis on the declarative, abstract level when teaching recursion considerably improved recursive program formulation [13], a result echoed by another researcher who suggested that a template emphasizing the practical use of recursion over the details of how recursion works showed promise in helping students to overcome comprehension difficulties [36]. Another author suggests that the analogy of delegation, that is, imagining recursion as a sort of task assigned by a boss to subordinates, is a productive approach for teaching students in majors outside of computer science [9]. Yet another analogy used to convey recursion is that of dominos tipping over, which was suggested by one author as a fruitful approach for any type of linear recursion problem [40].

An interesting line of research considered the relative difficulty of learning iteration versus learning recursion [26]. The author compared students who learned recursion first in a functional programming class to students who learned iteration first in a more traditional approach to CS1. He concluded that students learning recursion first were at least as skilled as students who learned iteration first, although he noted several caveats about the two populations and was hesitant to draw strong general conclusions because of confounding factors like motivation [26].

In another line of work relating recursion and iteration a researcher found that tail-recursive programming can be more effectively learned by applying a formal methodology for deriving the functions, although he cautioned that the approach should only be applied in a CS2 course due to the mathematics required [29].

Another area of student confusion is the development and understanding of base cases [18]. Their suggestions were to emphasize the declarative and abstract aspects of recursion, to be cautious in adapting or designing concrete models (such as the Russian Dolls model) so that they illustrate boundary values, and to make students explicitly aware of the issues in understanding base cases [18].

2.4 Mental Models

A large body of research in educational approaches to recursion is the study of student mental models. One of the earliest papers on the subject established that the mental model held by experts is the copies model where each process is capable of triggering a new instantiation of itself, but that novices most often held other incorrect models such as the looping model, null model, odd model, or magic model [22].

One paper focused on the relationship between cognitive learning styles and conceptual models of recursion [39] providing a particularly good survey of conceptual models for teaching recursion (including Russian Dolls, process tracing, stack simulation, mathematical induction, and structure template). The authors found that students with an abstract learning style performed better than those with concrete learning styles in learning recursion, that concrete conceptual models were better than abstract conceptual models in helping novice programmers to learn recursion, that abstract learners did not necessarily benefit more from abstract conceptual models, and that concrete learners did not necessarily benefit more from concrete conceptual models in learning recursion [39].

As an initial piece of a larger body of work, a group of researchers explored the types of mental models students develop about recursion, paying close attention to student understanding of the active flow (when control is passed to new instantiations) and passive flow (when control flows back from terminated recursive calls) in recursion [16]. They identified 8 student mental models, identifying which models were viable (led to correct understanding of recursion) and which were not and drawing connections between the recursion activities students perform and the mental models they develop. In related work [27], a questionnaire was developed to allow the assessment of student mental models of recursion, and four more general mental models consistent with previous work were suggested. Experimental results found that it was more fruitful to focus on declarative aspects of programming in helping students to develop correct mental models for recursion [27].

2.5 Connecting Pedagogy and Mental Models

There are a series of papers that draw together work in mental models and designing recursion problems. As described in the paragraph above, a group of researchers classified the types of mental models developed by students learning recursion [16],

considered the impact of introducing more complex recursive algorithms earlier [33], investigated the impact of a language switch from Scheme to Python [32], and considered the relationship between being able to trace a recursive function and write correct recursive solutions [34]. They concluded that the language switch had not had an impact, but that the changes to the lecture, labs, and tutorials placing a greater emphasis on algorithms that require an understanding of both the active and passive flow did improve students' ability to develop viable mental models for recursion [33]. They also recommended that instructors show students a variety of recursive problems to avoid instilling the belief that all recursive algorithms are similar in structure to mathematically-based algorithms [34].

One interesting study considered whether advanced students who had previously learned recursion in multiple classes were able to apply the technique to problems without being prompted to do so [12]. The author found that only a minority of students employed backward-reasoning approaches for problems made easier by that algorithmic technique, suggesting that recursion had not been assimilated sufficiently well to be retained.

One unusual study analyzed learners' discourse surrounding recursive phenomena as a way of understanding recursion through the students' eyes, discovering that learners see recursion in very different ways than educators and experts [25].

2.6 Non-traditional Populations

Several authors focus on projects that address non-traditional populations. K-12 students are one target audience, with one study focusing on teaching students aged 11-14 in an extracurricular program in the university setting [17]. Other authors consider teaching end-user programmers, that is, coders who do not program as the main function of their job [9].

2.7 Motivation

Given the widely acknowledged difficulty of learning recursion, it is surprising that few researchers consider the issue of student motivation. Based on the results of their study of students aged 11-14 in an extracurricular program teaching recursion, Gunion and her collaborators suggested that recursion activities can improve student interest in computing [17]. As a footnote to a study on whether iteration or recursion first made a difference in student comprehension, one researcher noted that it was difficult to draw the conclusion that teaching recursion first before iteration led to deeper learning because the motivational levels between the two populations studied may have differed [26].

Motivation for learning recursion has been directly considered by a group of researchers involved in the Game2Learn project, who as part of their work developed EleMental: The Recurrence [5], a game for teaching recursion. In the game students complete three recursion puzzles on a binary tree helped by Ele, a programmable avatar. The study showed that students achieved statistically significant learning gains while playing the game, and that most of the students were enthusiastic about learning with the game, and about the possibility of using more such games in learning complex computing topics [5].

Motivation was also an important consideration for an author who detailed three graphical problems that are more easily solved using recursion than using iteration [37]. He hypothesized that showing students in CS1 or CS2, who have yet to see trees or sorting algorithms, problems for which recursion is a valuable problem-solving tool was likely to be motivating for them. The recursive solutions to these problems demonstrated that the approach could be both clear and efficient.

3. A MOTIVATIONAL APPROACH

As seen in the summary above the papers addressing student motivation represent a small fraction of the body of work on teaching recursion. It can be argued that effective pedagogy should take precedence over motivation for students learning recursion, who are, after all, typically more advanced in their studies. But even a study focused on other aspects could consider motivation as one of the outcomes of its interventions, and this appears to not be the case for most researchers.

A workshop presented at an information technology education conference considered an approach that has significant motivational aspects [35]. In the text from which this approach is taken [28] the chapter on recursion appears immediately before the chapter on web application development. The recursion chapter begins with a series of simple functions that operate on integers, and a discussion of recursive function calls and the stack is presented next. The following section has multiple examples of recursive functions including another pattern printing problem and a function that prints Koch's curve. The section concludes with a function that simulates a virus-scanning program, introducing the Python os module. A later section considers searching, describing first linear search and then binary search. A chapter on web application development and web searching immediately follows the recursion chapter and discusses the Python WWW API where three important modules are discussed. The module urllib.request allows HTML files to be opened in much the same way that files are opened. The module html.parser provides a parent class HTMLParser that can be overridden to parse HTML files in various ways. The final module is urllib.parse which contains a method urljoin that allows a programmer to construct absolute URLs from relative URLs found in web pages. With all of the pieces in place the final section is a case study of the development of a web crawler. The chapter concludes with a discussion of how to do web page analysis using ideas about text processing introduced in earlier chapters.

This approach to teaching recursion employs multiple best practices seen in the literature. A multitude of recursion problems of various types are considered, including visually-oriented examples [10, 15] including printing functions and Koch's snowflake, problems that lend themselves to tail-recursive solutions [29, 31] such as several of the printing functions and factorial, combinatorial problems like Fibonacci and combinations [31], problems that fail to utilize recursion well such as Fibonacci, and problems for which recursion allows for easier development of efficient recursive solutions [37] such as exponentiation. As is common in most modern textbooks the approach is high-level with relatively little time spent discussing the mechanics of the stack and activation records [13]. The various problems discussed lend themselves to a variety of base cases [18], including some for which the function does nothing at all. There is no concrete model of recursion used in the chapter [39], with the explanation instead relying on a wealth of different examples to illustrate various aspects of the development of recursive functions. The

examples presented require the use of passive and active flow during recursion in multiple ways [33, 34], with factorial, pattern printing, Koch's snowflake, and the virus scanner all demonstrating various approaches to decomposing and reconstructing solutions using recursion. The text does not employ all of the ideas found in the literature, which to be fair, would be difficult given that several of them are incompatible. For example, there is no discussion of recursive data structures [2, 3, 14]. Trees [24] or arbitrarily nested lists are not used as examples, and recursion trees [20] are not provided as a part of the explanations. Many of the examples have iterative solutions that are equally simple as the recursive solutions [37]. Tracing recursive functions [34] is not a focus of the chapter.

This approach is particularly appealing for information technology students or for computing students with an interest in web development. The use of recursion is very natural in certain contexts in web development, and students who understand the utility of an approach are more likely to spend the time necessary to reach the all-important 'aha' moment that comes with mastery of that technique.

4. CONCLUSION

Recursion is a particularly well-studied problem in the computing education literature. Articles dating from the 1980s have considered various aspects of teaching recursion including novel pedagogical approaches, fruitful and illuminating problems, understanding and influencing students' mental models of recursion, the relationship between non-traditional populations and recursion, and the impact of recursion on interest in computing. Results found in the literature were summarized in this article, drawing connections between related lines of work.

Interestingly and despite the demonstrated relationship between student motivation and learning to program [4], very little attention is paid to the issue of student motivation for learning recursion. One possible explanation for this could be that students learning recursion are typically more advanced in their studies, making motivation less of an issue. But this is not the case for some branches of recursion research, such those interested in spurring interest in computing or in reaching non-traditional populations. This gap in the recursion literature is surprising. We briefly described an approach to using web development as a motivator for recursion, but there are no doubt many other ways students can be encouraged to tackle the complex and difficult subject. Finding effective ways to motivate students to learn recursion is clearly an open problem and should be addressed by computing education researchers.

5. ACKNOWLEDGEMENTS

We thank Ljubomir Perković for his support in writing this article and for his comments on earlier drafts. We also thank André Berthiaume for his feedback on earlier drafts.

6. REFERENCES

[1] Ben-Ari, M. 1997. Recursion: From Drama to Program. *Computer Science Education*, 11:3, pp. 9 – 12.

[2] Bloch, S. 2003. Teaching Linked Lists and Recursion Without Conditionals or Null. *Journal of Computing Sciences in Colleges*, 18:5, pp. 96-108.

[3] Bruce, K.B., Danyluk, A., and Murtagh, T. 2005. Why Structural Recursion Should Be Taught Before Arrays in CS 1. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 2005).

[4] Carbone, A., Hurst, J., Mitchell, I., and Gunstone, D. 2009. An Exploration of Internal Factors Influencing Student Learning of Programming. In *Proceedings of the 11th Australasian Computing Education Conference*, (Wellington, New Zealand, January 2009).

[5] Chaffin, A., Doran, K., Hicks, D., and Barnes, T. 2009. Experimental Evaluation of Teaching Recursion in a Video Game. In Proceedings of the *2009 ACM SIGGRAPH Symposium on Video Games* (New Orleans, Louisiana, August 2009).

[6] *Computer Science Curricula 2013: Ironman Draft (Version 1.0)* February 2013. The Joint Task Force on Computing Curricula, Association for Computing Machinery, IEEE-Computer Society, CS2013 Steering Committee, accessed August 2013.

[7] CS Unplugged, http://csunplugged.org/, accessed August 2013.

[8] Dale, N.B. 2006. Most Difficult Topics in CS1: Results on an Online Survey of Educators. *ACM SIGCSE Bulletin*, 38:2, pp. 49 – 53.

[9] Edgington, E. 2007. Teaching and Viewing Recursion as Delegation. *Journal of Computing Sciences in Colleges*, 23:1, pp. 241-246.

[10] Elenbogen, B.S. and O'Kennon, M.R. 1988. Teaching Recursion Using Fractals in Prolog. In *Proceedings of the 19th SIGCSE Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA, February 1988).

[11] Ford, G. 1984. An Implementation-Independent Approach to Teaching Recursion. In *Proceedings of the 15th SIGCSE Technical Symposium on Computer Science Education* (Philadelphia, Pennsylvania, USA, February 1984).

[12] Ginat, D. 2004. Do Senior CS Students Capitalize on Recursion? In *Proceedings of 9th Annual Conference on Innovation and Technology in Computer Science Education* (Leeds, United Kingdom, June 2004).

[13] Ginat, D. and Shifroni, E. 1999. Teaching Recursion in a Procedural Environment – How much should we emphasize the Computing Model? In *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, March 1999).

[14] Goldwasser, M.H. and Letscher, D. 2007. Teaching Strategies for Reinforcing Structural Recursion with Lists. *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA Educators' Symposium) (Montreal, Quebec, Canada, October 2007).

[15] Gordon, A. 2006. Teaching Recursion Using Recursively-Generated Geometric Designs. *Journal of Computing Sciences in Colleges*, 22:1, pp. 124-130.

[16] Götschi, T., Sanders, I., and Galpin, V. 2003. Mental Models of Recursion. In *Proceedings of the 34th SIGCSE*

Technical Symposium on Computer Science Education (Reno, Nevada, USA, February 2003).

[17] Gunion, K., Mildford, T., and Stege, U. 2009. Curing Recursion Aversion. In *Proceedings of 14th Annual Conference on Innovation and Technology in Computer Science Education* (Paris, France, July 2009).

[18] Haberman, B. and Averbuch, H. 2002. The Case of Base Cases: Why are They so Difficult to Recognize? Student Difficulties with Recursion. In *Proceedings of 7th Annual Conference on Innovation and Technology in Computer Science Education* (Aarhus, Denmark, June 2002).

[19] Henderson, P.B. and Romero, F. J. 1989. Teaching Recursion as a Problem-Solving Tool Using Standard ML. In *Proceedings of the 20th SIGCSE Technical Symposium on Computer Science Education* (Louisville, Kentucky, USA, February 1989).

[20] Hsin, W.J. 2008. Teaching Recursion Using Recursion Graphs. *Journal of Computing Sciences in Colleges*, 23:4, pp. 217-222.

[21] Information Technology 2008. Curriculum Guidelines for Undergraduate Degree Programs in Information Technology. Association for Computing Machinery and IEEE Computer Society. http://www.acm.org//education/curricula/IT2008%20Curriculum.pdf, accessed August 2013.

[22] Kahney, H. 1983. What Do Novice Programmers Know About Recursion. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (Boston, Massachusetts, USA, December 1983).

[23] Kann, C., Lindeman, R.W., and Heller, R. 1997. Integrating Algorithm Animation into a Learning Environment. *Computers & Education*, 28:4, pp. 223 – 228.

[24] Kruse, R.L. 1982. On Teaching Recursion. In *Proceedings of the 13th SIGCSE Technical Symposium on Computer Science Education* (Indianapolis, Indiana, USA, February 1982).

[25] Levy, D. 2001. Insights and Conflicts in Discussing Recursion: A Case Study. *Computer Science Education*, 11:4, pp. 305-322.

[26] Mirolo, C. 2012. Is Iteration Really Easier to Learn than Recursion for CS1 Students? In *Proceedings of International Computing Education Research Workshop* (Auckland, New Zealand, September 2012).

[27] Mirolo, C. 2010. Learning (through) Recursion: A Multidimensional Analysis of the Competences Achieved by CS1 Students. In *Proceedings of 15th Annual Conference on Innovation and Technology in Computer Science Education* (Ankara, Turkey, July 2010).

[28] Perković, L. 2012. *Introduction to Computing using Python: An Application Development Focus*. John Wiley & Sons.

[29] Rubio-Sánchez, M. 2010. Tail Recursive Programming by Applying Generalization. In *Proceedings of 15th Annual Conference on Innovation and Technology in Computer Science Education* (Bilkent, Ankara, Turkey, June 2010).

[30] Rubio-Sánchez, M. 2008. An Introduction to Problem Equivalence with Combinatorics. In *Proceedings of 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain, June/July 2008).

[31] Rubio-Sánchez, M. and Hernán-Losada, I. 2007. Exploring Recursion with Fibonacci Numbers. In *Proceedings of 12th Annual Conference on Innovation and Technology in Computer Science Education* (Dundee, Scotland, United Kingdom, June 2007).

[32] Sanders, I. and Galpin, V. 2007. Students' Mental Models of Recursion at Wits. In *Proceedings of 12th Annual Conference on Innovation and Technology in Computer Science Education* (Dundee, Scotland, United Kingdom, June 2007).

[33] Sanders, I., Galpin, V., and Götschi, T. 2006. Mental Models of Recursion Revisited. In *Proceedings of 11th Annual Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy, June 2006).

[34] Scholtz, T. and Sanders, I. 2010. Mental Models of Recursion: Investigating Students' Understanding of Recursion. In *Proceedings of 15th Annual Conference on Innovation and Technology in Computer Science Education* (Ankara, Turkey, July 2010).

[35] Settle, A. 2013. Reaching the 'Aha!' Moment: Web Development as a Motivator for Recursion. In *Proceedings of the 14th Annual Conference in Information Technology Education* (Orlando, Florida, October 2013).

[36] Sooriamurthi, R. 2001. Problems in Comprehending Recursion and Suggested Solutions. In *Proceedings of 6th Annual Conference on Innovation and Technology in Computer Science Education* (Canterbury, United Kingdom, June 2001).

[37] Stephenson, B. 2009. Using Graphical Examples to Motivate the Study of Recursion. *Journal of Computing Sciences in Colleges*, 25:1, pp. 42-50.

[38] Wirth, M. 2008. Introducing Recursion by Parking Cars. *ACM SIGCSE Bulletin*, 40:4, pp. 52-55.

[39] Wu, C., Dale, N.B., and Bethel, L.J. 1998. Conceptual Models and Cognitive Learning Styles in Teaching Recursion. In *Proceedings of the 19th SIGCSE Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA, February 1988).

[40] Yang, F-J. 2008. Another Outlook on Linear Recursion. *ACM SIGCSE Bulletin*, 40:4, pp. 38-41.