

Literature Review

Visual Maps to navigate Non-Linear Learning Environments

María Isabel Correa

[mic2130@tc.columbia.edu](mailto:mic2130@tc.columbia.edu)

MSTU 4088 Introduction to Educational Technology and the Learning Sciences

Teachers College, Columbia University

## **Abstract**

This paper provides a review of previous research on the use of visual maps as an orientation tool for students by enhancing visual-spatial and metacognitive reasoning when learning within non-linear environments. Inspired by specific design features of developed studies on learning maps, I propose a mapping tool to guide students in learning computer programming by aiding the navigation through interrelated and non-linear sets of codes.

## **Problem Statement**

Information has gone from scarce to superabundant and increasingly complex due to advances in technology and accessibility (Shute, 2011). With the aim of preparing future generations with learning experiences that equip them to interact effectively with information, new learning paradigms have been integrated into schools such as problem-based learning, inquiry-based learning, and expeditionary learning. These learning experiences require students to actively retrieve information in non-linear ways and deal with complex sets of data from the web and other multimedia sources to solve open-ended problems. These resources are usually characterized by their network-like structure of inter-related nodes with stored information in various media forms, such as texts, graphics, videos, and sounds (Robberecht, 2007). Students are required to actively navigate through these non-linear environments and interact with data in multiple ways (Opfermann, Scheiter, Gerjets, & Schmeck, 2013; Oliver & Herrington, 1995). The flexibility of these learning experiences can foster engaging, active, and constructive learning but also poses significant challenges in how we prepare students to navigate through complex information (Opfermann et al., 2013). The main problem is that non-linear learning can disorient students in their learning process (Lee & Baylor, 2006; Lin, Hmelo, Kinzer, & Secules,

1999; Oliver & Herrington, 1995). According to Conklin (as cited in Lee & Baylor, 2006), disorientation is the “tendency to lose the sense of direction and location in non-linear environments.” This disorientation can limit instructional effects by distracting students in the learning process and requiring a longer time to complete tasks (Opfermann et al., 2013). This problem can negatively affect students’ learning outcomes and also prevents teachers from incorporating non-linear learning experiences as part of their lessons.

Previous research on nonlinear learning environments has identified two main internal causes for students’ disorientation: 1) students’ lack of visual-spatial skills needed to understand the structure of the nonlinear information (Baylor, 2001; Lee & Baylor 2006), and 2) students’ lack of metacognitive skills needed to recognize their current positions and next directions of their learning processes within the structure of information (Dunning, Heath, & Suls, 2004; Mazumder & Finney, 2012; Yeung & Summerfield, 2012). There are also other non-internal causes for this problem, such as design problems of the instructional materials (Oliver & Herrington, 1995; Robberecht, 2007; Tufte, 1990, 1997, 2000, 2006) or lack of professional training for teachers on how to implement these non-linear learning environments (Inan & Lowther, 2010), but for the purpose of this paper, we will only focus on the internal causes.

Ideally, well-oriented learners should be able to recognize their current positions and next directions (Lee & Baylor, 2006) by engaging actively in visual-spatial and metacognitive processing. On one side, *visual-spatial thinking* can be explained as the “ability to perceive the whole picture from only the parts” which is particularly relevant for information processing (Baylor, 2001). To explain this, David Hyerle (1996) makes the analogy between dealing with complex information and navigating through a forest, where you are required to see, simultaneously, the forest and the trees to have the macro-vision of the whole subject as well as

the complex micro-vision of interrelated details. On the other side, *metacognition* is commonly defined as the ability to understand and monitor one's own thoughts and the assumptions and implications of one's activities (Favell, 1987). It could be understood as the ability to "drive" our own brain, where you are required to know where you currently are and the alternative routes to reach your objectives. Similarly, Brown (as cited in Lee & Baylor, 2006) depicts metacognition as awareness and control or regulation about cognition and learning process. When learning within nonlinear environments both aspects of metacognition are required: awareness of your current level of understanding of the information and control to take navigation decisions to improve it (Lin, 2001). This ability allows students to take advantage of information to effectively solve problems (Lin, Hmelo, Kinzer, & Secules, 1999) and avoid overcommitting time and resources to decisions that are based on unreliable evidence (Yeung & Summerfield, 2012). Hyerle and Alper (2011) said that oriented learners navigate through nonlinear environments in a similar way that a person equipped with a GPS would do through a forest. The GPS provides a map to visualize the whole forest from the top while the person can still see the trees. The GPS also provides the current location of the person, as well as points of reference and roads, in order to take decisions and navigate.

Previous studies have suggested different ways to support students' orientation in nonlinear learning environments by fostering visual-spatial and metacognitive reasoning. Many studies have focused on providing support for metacognition because it has been proven that despite the benefits of metacognition as a tool to navigate through complex information, students do not spontaneously engage in metacognitive thinking unless they are explicitly encouraged to do so (Lin, 2001; Bannert & Mengelkamp, 2013). It is not surprising that many of them have suggested the idea of using maps to guide students' orientation. In the following section, we will

review different studies on how maps can support visual-spatial and metacognitive reasoning and their impact on learning in nonlinear environments. Each study will be explained with a brief description, an overview of the finding, and an analysis of strengths and weaknesses.

## **Literature Review**

### **Open Learner Models**

Bull and Kay (2013) argued that Open Learner Models (OLM) could support metacognitive activities. Open Learner Models are digital graphic models that capture the learner's present understanding of a domain based on their interaction with an interface. Using user's data such as navigation choices, answers to questions, problem-solving attempts, and time on task, they generate a representation of the learner's current state of understanding in relation to a whole model of the domain or subject of study. Learner models are commonly a backstage part of intelligent tutoring systems, used to automatically generate instruction adapted to the learner's state of understanding. The authors' argument is that tutoring systems that leave learner models available for the user could support metacognitive reasoning by confronting the student with a representation of their understanding. To explain this, the authors classified and qualitatively analyzed an array of OLMs with the aim of identifying design principles and features for supporting metacognitive activities.

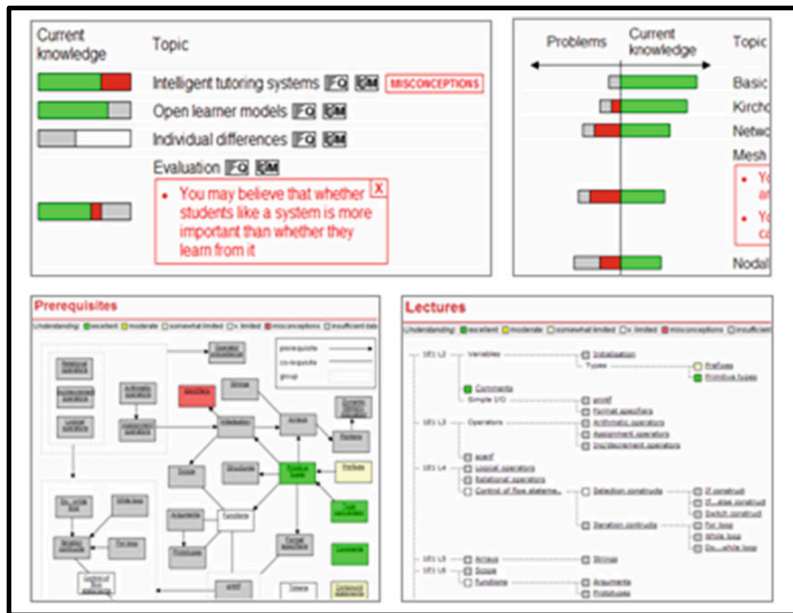


Figure 1 Example of an Open Learner Model from an Intelligent Tutoring System

*Findings.* Across all the variety of OLMs, Bull and Kay (2013) identified different design models. Some models compared the individual level understanding with points of reference such as the whole domain, peers' current understanding, or tutor's expectations. Other features aided students' decision making by providing a clear visualization of their starting point. Others provided the means to alternatively visualize the big picture of the whole learning domain or zoom to focus on content details. Some were designed to help the learner identify any problematic issues and then find solutions on their own. Worth mentioning are OLMs that allowed user interaction to negotiate their levels of knowledge and increase learner control.

*Analysis.* The study of Open Learner Models provides valuable insights on how designed graphic representations can mirror learners' understanding with the aim of supporting metacognitive activities. Nevertheless, as the authors mentioned, it is now necessary to scientifically assess and interpret metacognitive activities using OLMs (Bull & Kay, 2013). This study can potentially compare navigation data of students with and without access to OLMs to

analyze their metacognitive behavior. Another shortcoming of many OLMs is the lack of opportunities for student's interaction. Finally, OLMs are not a common feature in web pages and tutoring systems, which make them an inaccessible metacognitive tool.

## Metacognitive Maps

Lee and Baylor (2006) proposed Metacognitive Maps to support metacognition in web-based learning processes. A metacognitive map is a visual interface tool designed to aid four main metacognitive skills: planning, monitoring, evaluating, and revising. It is composed of three parts: global, local tracking map, and planning space. The global map is used to show the structure of the entire learning content, and to guide students to plan their activities effectively. The local tracking map is used by learners to check what they have already done, and decide what they need to do next. Finally, the planning space provides a mechanism to support learner's premeditated planning to the learning tasks (i.e.: define learning goals, strategies, and expected time).

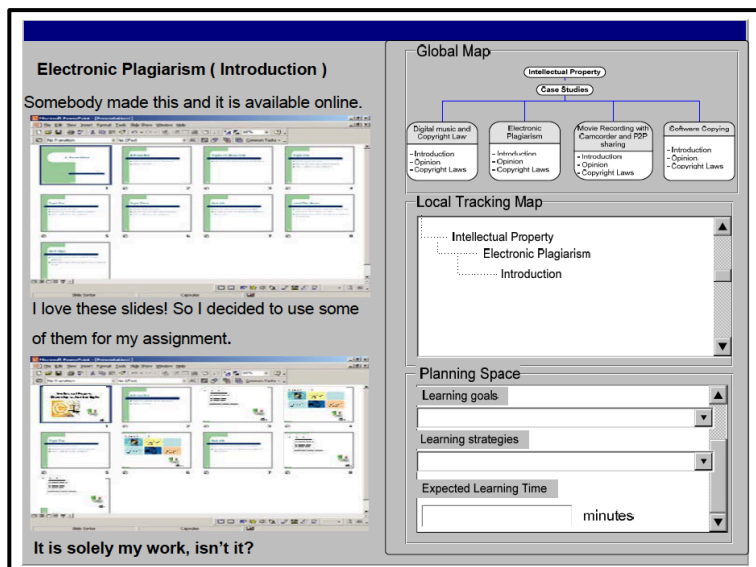


Figure 2. Screenshot of the Metacognitive Map software

*Findings.* The authors proposed and described this specific design to support metacognitive reasoning within web-based learning environments (Lee & Baylor, 2006). The visual representation allows user interaction, which places in students the responsibility of tracking their progress and completing the planned goals to induce metacognitive reasoning.

*Analysis.* One strength of this design is its foundation on theory on metacognitive and visual-spatial thinking. It provides concrete features to support the key skills of metacognition and it also provides support to visual-spatial orientation through the global and local tracking maps. It is valuable that students have an active role in the interaction with the platform. Nevertheless, as the authors pointed out, it is necessary to empirically evaluate how effective it is supporting metacognition and learning (Lee & Baylor, 2006). Regarding accessibility, it is also restricted to web pages or software that include this feature, which is generally rare.

### **Thinking Maps®**

David Heyrle (1996; 2011) proposed Thinking Maps as a visual-verbal language for thinking and communicating with the aim of supporting students' capacity to transfer knowledge, reflect upon, and improve their thinking abilities. This language consists of a series of eight Thinking Maps or visual-verbal models that represent the following eight cognitive operations: seeking context, describing attributes, comparing and contrasting, inducting and deducting, identifying part-whole relationships, seeking causes and effects, and making analogies. Besides the eight models, there is an additional map called the "metacognitive frame", that may be drawn around any of the maps as a space for identifying and sharing one's frame of reference for the information used in the Thinking Map (i.e.: personal stories, culture, belief systems, influences, etc.). Thinking Maps, by mirroring cognitive patterns, help students to



become conscious of their mental operations and transfer them into any learning environment. Thinking Maps can be integrated into any subject domain and ideally must be introduced in schools as a common visual language for thinking and learning across the whole learning community in order to generate an orchestration of interdependent thinking process. This way, Thinking Maps becomes a tool to tap cognitive patterns, make them visible, mediate them, assess them and enhance them. The model also evolved to software (Hyerle & Gray Matter Software, 2007) that provides tools for students to generate the same Thinking Maps within a digital platform. The software is designed for both teachers and students through a three-window approach: a window for lesson planning for the teacher, a window for the generation of Thinking Maps for students, and a window for students to transfer their maps into writing. Teachers can also generate Thinking Maps of the lessons, create plans and assessments, or capture evidence of progress.

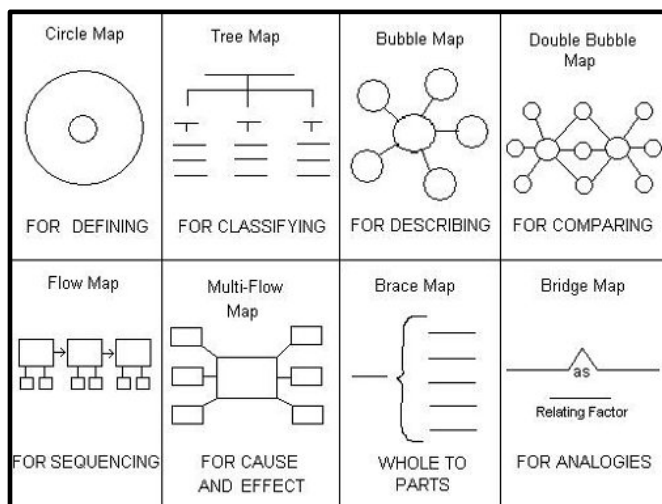


Figure 3. The set of eight Thinking Map basic units of language

*Findings.* Since the publication of Thinking Maps in 1988, numerous studies have been conducted to demonstrate their impact in learning. Researchers found that Thinking Maps can

improve students' ability to see relationships between main ideas and supporting details and that this in turn led to higher scores on reading and writing tests (Cronin et al. as cited in Hyerle, 1995). Similarly, studies conducted in schools in the US demonstrated significant improvements in standardized tests over successive years of implementation of Thinking Maps with many schools doubling their scores (Hyerle, 1996). Also a shift from passive to interactive learning fueled by students' motivation was observed. Thinking Maps have been especially successful in schools that have implemented them across all disciplines, throughout grade levels, and over several years. Consequently, schools generate a "common thinking process, vocabulary and visual language" (Hyerle, 1995). According to teachers, the maps have effectively helped students become independent and reflective learners. Students become more aware of how the maps display what they learned which turns in metacognitive dialogue among the learning community (Hyerle, 1995).

*Analysis.* Thinking Maps provide a dynamic set of visual tools to support effective navigation through non-linear environments. Moreover, when implemented across learning communities, they provide a common language to translate students' thinking process and enhance metacognitive activities. Even if it is necessary to provide strong professional training for teachers on Thinking Maps, the entry barriers are lower as opposed to the other tools where specific software is required to generate maps. In this case, students build their own maps, which contribute to students' construction of learning and to the generation of a graphic representation to support metacognitive activities such as self-explanation, self-evaluation, decision-making, and planning. Furthermore, Thinking Maps are basic structures or primitives which allows multiple possibilities of combinations, expansions, and applications making them an open system for learning.

## Representation of problem solving stages

Loksa and colleges (2016) developed an approach to enhance metacognitive skills in the instruction of problem solving within the context of a computer programming camp. They proposed a series of interventions to help students recognize, evaluate, and refine their problem solving strategies. One relevant intervention from the perspective of our analysis was a visual representation of problem solving stages to help learners monitor their understanding.

Concretely, it consisted in a physical handout with a diagram map that detailed programming problem solving stages and encourages learners to track their current stage and plan next actions with the aid of reflective prompts. The diagram depicted six key stages for problem solving: interpret problem prompt, search for analogous problems, search for solutions, evaluate a potential solution, implement a solution, and evaluate implemented solution. Besides that representation, they also provided other strategies to enhance metacognition such as explicit instruction of the goals and activities, and context-sensitive prompts.



Figure 4. The paper handout to track problem solving stages

*Findings.* To measure the effects of the mentioned interventions the authors compared a traditional version of a web development camp (control) with an experimental version of the

same camp, which included the described interventions (Loksa et al., 2016). To assess metacognitive awareness, campers completed an end-of-day survey asking to reflect on a difficult task and their attempt to solve it. They found that campers in the experimental group were significantly more likely to write an explicit description of a problem solving strategy. This demonstrates the effectiveness of the diagram together with the other interventions to enhance metacognition.

*Analysis.* This study provides interesting insights on how visual representations can be complemented with instructional strategies to enhance certain behaviors such as monitoring, revising and evaluating. Complementary interventions such as prompting are particularly important in metacognition since it is an activity that does not happens spontaneously. As a result, the experiment shows how the interventions working together contribute to self-awareness during a problem solving process. Nevertheless, it is not possible to see the relative contribution of the different interventions to this outcome. For instance, there is no evidence of the influence of the diagram itself on the observed metacognitive outcomes. Furthermore, it is not possible to know if the particular visual form of the representation contributed to the outcomes.

## **Solution proposal**

### **SeeSe, programming map to see sequences of code in JavaScript**

In the previous section, I reviewed some examples of initiatives to enhance visual-spatial thinking and metacognition using visual maps of the learning process in order to guide students within the context of non-linear environments. We can conclude that maps could be a powerful tool for students to navigate through learning environments where information is complex and interrelated. In a similar way, and following the ideas of Loksa and colleagues, maps may aid the learning process of other non-linear reasoning tasks such as learning computer programming, which is rapidly becoming a 21<sup>st</sup> century literacy. Recent studies on software engineering expertise have found that the most successful engineers are systematic and self-aware (Lee & Ko, 2015). Furthermore, studies have demonstrated that within the context of learning computer programming students' development of metacognitive strategies had a significant short-term and long-term effect on students' learning outcomes (Volet, 1991; Loksa et al., 2016). According to Loksa and colleagues (2016), learning programming involves more than merely knowing a language's syntax and semantic; it requires an "iterative process of refining mental representations of computational problems and solutions, and expressing those representations as code." In other words, students need to translate their mental representations of computational problems into a new language that is the language of the computer or code. Using maps to tap into those mental representations may help learners monitor this complex process. Based on this idea, SeeSe map was developed as a tool for students to guide them in the process of building an assertive mental representation of computational code by creating a visual representation of it. SeeSe is a tool for learning basic aspects of JavaScript in computer programming with a focus on teaching the structural aspects of the code by making thinking processes visible. It is worth

mentioning that while Loksa's and his colleagues intervention was focused in scaffold problem solving thinking this tool focuses on support structures and the sequencing thinking characteristic of computer programming.

The main objective of this tool is teaching students *how to think* when programming, specifically by fostering their visual-spatial and metacognitive skills. It is grounded in the revised theory that generating maps to make thinking processes visible facilitates student reflection and orientation (Bull & Kay, 2013; Hyerle, 1995; Lee & Baylor, 2006). The main difference with other technologies, such as Scratch or Blockly, is that those tools provides a visual interface on top of code, where students can program things without writing code at all. As a result, most learners use these tools to create content, possibly avoiding coding all together (Scaffidi and Myers, as cited in Loksa et al, 2016). SeeSe on the other side is designed for students to embrace the complexity of real code by exercising the ability to build mental representations of it and translate them into visual structures that can scaffold the understanding of code syntax.

JavaScript is programming language commonly used in web development to add dynamic and interactive elements to websites. A JavaScript code is composed primarily of values. These values can be fixed such as numbers or strings (text) or variables that are used to store data values. Basically, a JavaScript code is list of instructions to be executed by the computer in order to compute values. Those instructions are called statements that, in general, are a combination of values, variables, and operators, which compute a value. This computation is called evaluation. JavaScript code can be understood as a unique sequence of inter-related instructions meant to evaluate values. That means that behind the complexity of the code there is a logic structure where one statement leads to the next statement in a predetermined order. The understanding of this sequential logic of JavaScript code is the key to knowing how values are

going to be evaluated by the computer. This understanding can be challenging for students, because each statement of the code generates new evaluations of variables previously defined, making difficult to keep track of the values throughout the code. In other words, students may get disoriented in the sequence of code. For this reason, a map that visually represents the sequence of statement and the stages of the values may aid students in this process.

Maps and diagrams can be understood as visual languages. According to Robert E. Horn (1998), visual languages are defined as an integration of words and shapes into a single communication unit. In the same way, a SeeSe map, can be understand as a visual language and its components can be classified into vocabulary and grammar. The vocabulary of this language consists in shapes with the form of containers that store values written as words or numbers. In addition, the grammar is a set of rules to put the vocabulary together in order to make visible the sequential aspect of the code (see figure 5). Using these two components students can translate a code written in JavaScript into SeeSe visual language. The result will be a map representation that may reflect students' understanding of the sequence of statements and their evaluation of values through the code (see figure 6). Following the design principle of the Open Learner Model and the Metacognitive Maps, confronting the learners with a representation of their understanding can enhance metacognitive activity and visual-spatial skills.

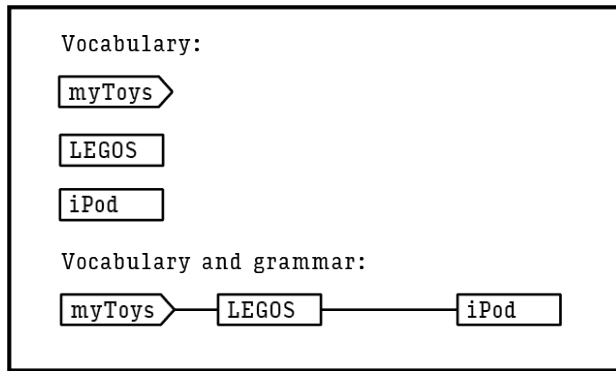


Figure 5. The vocabulary is the set of container shapes to represent the variables and values. The grammar is a set of rules about how to organize the shapes in order to convey the sequential aspect of the code.

Both the vocabulary and the grammar of SeeSe were based on design principles of previous research. The vocabulary was inspired by DiBiase’s (1995) experiment of scaffolding the understanding of abstract mathematical functions using objectification through visual or physical representations. In a similar way SeeSe represents abstract concepts such as variables and values as boxes to store different things. The grammar was influenced by the work of David Hyerle (2011) and the Thinking Maps, specifically the flow diagram. This map was first developed by the pioneer of modern computer sciences, John Von Neumann (1903 – 1957), as a way to visually represent computer instructions for people understanding (Horn, 1998). Since then, the flow map is used to show a flow or set of sequential processes in a system where one event leads to another. For instance, flow diagrams are used to represent timelines, cycles, programs, etc. This map provides a suitable structure for learners to see the logical sequence that relays behind the inter-related statements of the code. The flow map of SeeSe has particular rules about how to place the containers in order to reveal the sequence of the code and to keep track of the successive evaluations.



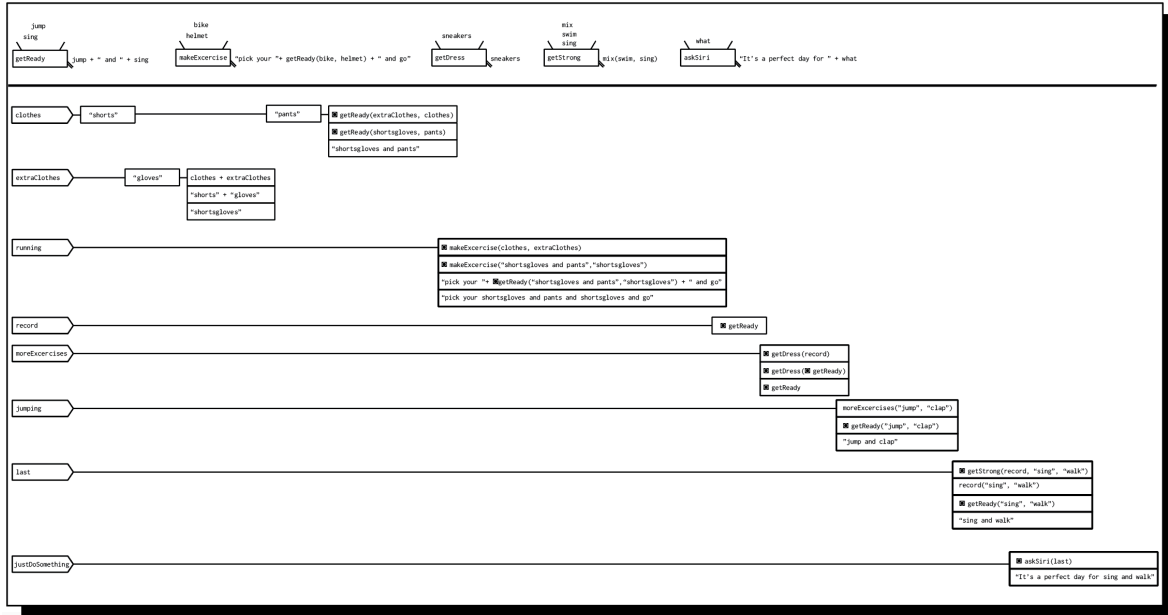


Figure 6. A SeeSe map of a JavaScript code

As any new language, SeeSe needs to be taught and practiced in order to reach its full potential as a tool for metacognition and visual-spatial thinking. For this purpose a curriculum was designed for teachers to train learners in the acquisition of the visual language and to support metacognitive reasoning using the resulting maps (see Appendix). The curriculum was developed for adult learners with no previous experience in JavaScript. It is composed of explanations on how to teach the vocabulary and the grammar, exercises to translate code from JavaScript to SeeSe and vice versa, and self-reflective prompts and peer-evaluations to encourage metacognitive activity. For the moment, students can generate a SeeSe map using analog tools as paper and pencil to draw the map, but this process can be easily transferred into software designed to facilitate the creation of SeeSe maps. It is worth to highlight that SeeSe maps, as any other visual representation, may not encourage metacognitive activity if it is not complemented with instruction for this purpose. For this reason, a future development of software could incorporate reflective prompts to reach the full potential of the tool to support revising, evaluating, and monitoring.

While the development of SeeSe and its curriculum is theoretically grounded, it is necessary to empirically assess the effects of this support tool to understand how it impacts the learning process. The primarily expected outcomes of the implementation of the curriculum are an increment of visual-spatial skills and metacognitive skills in students. Visual-spatial skills must be reflected in their ability to see simultaneously the whole picture and the details. In computer programming, this can be interpreted as the ability to see how each statement works individually and how it is tied to the whole structure of code in order to compute values. Students' elaboration of a SeeSe map, might boost their visual-spatial skills by providing a representation of the whole structure of the sequence where they can see the whole picture of the code as well as the details of how the values are computed statement by statement. In addition, metacognitive activity in computer programming can be understood as students' ability to monitor their understanding of the sequence of code and revise their evaluation process. Making SeeSe maps can also foster metacognitive thinking by providing a visual language to translate their understanding of the sequence and a means to evaluate each stage using the graphic containers. The resulting visualization may support several metacognitive strategies relevant for computer programming such as self-explanation, de-bugging, and revising. In addition, the elaboration of SeeSe maps within a community of learning may boost collaboration and reflective dialogue as a consequence of allowing all students to speak the same visual language to convey their thinking process. SeeSe map could be a rich tool for teachers and classmates, to access to each other way of representing the code, which can facilitate dialogue and assessment.

Because the purpose of SeeSe maps is to improve the way students think about computer programming, another general expected outcome is a robust understanding of basic principles of JavaScript language. Specifically they may have a better conception of the sequential structure of

the code and how each statement is inter-related to others in order to compute values. In addition to all these aspects, it can be evaluated if learning basic concepts of JavaScript using SeeSe may impact how students interpret more complex or advanced sets of code. Finally, the possibility of opening the scope of this visual language to other aspects of JavaScript such as functions is subject of future research.

### **Conclusion**

As we move to more complex systems of information fueled by technology, we need to help students navigate through them. Furthermore, we are faced with the challenge of preparing future generations to build and develop those complex systems by integrating computer science at schools. We must provide students with flexible tools capable of making complexity accessible and comprehensible. Visual languages such as maps and diagrams have the potential to depict and communicate complex information in precise and efficient forms. Integrating visual languages into schools could enable students to think about complex processes and engage in skillful metacognitive analysis without losing the sense of direction to reach their learning goals. Regardless of the strong theoretical foundations of this idea, more empirical research is required to demonstrate the potential of visual tools to improve learning outcomes in today's non-linear learning environments.

## Bibliography

- Bannert, M. & Mengelkamp, C. (2013). Scaffolding hypermedia learning through metacognitive prompts. In Azevedo, R., & Alevan, V. (Eds.). (2013). *International Handbook of Metacognition and Learning Technologies* (Vol. 28). New York, NY: Springer New York.  
<https://doi.org/10.1007/978-1-4419-5546-3>
- Baylor, A. L. (2001). Perceived disorientation and incidental learning in a web-based environment: Internal and external factors. *Journal of Educational Multimedia and Hypermedia*, 10(3), 227–252.
- Bull, S. & Kay, J. (2013). Scaffolding Hypermedia Learning Through Metacognitive Prompts. In Azevedo, R., & Alevan, V. (Eds.). (2013). *International Handbook of Metacognition and Learning Technologies* (Vol. 28). New York, NY: Springer New York.  
<https://doi.org/10.1007/978-1-4419-5546-3>
- DiBiase, J., & Eisenberg, M. (1995). Mental imagery in the teaching of functions. *National Science Foundation*, Arlington, VA. Retrieved from <http://eric.ed.gov/?id=ED392425>
- Dunning, D., Heath, C., & Suls, J. M. (2004). Flawed self-assessment implications for health, education, and the workplace. *Psychological Science in the Public Interest*, 5(3), 69–106.
- Flavell, J. H. (1987). *Speculations about the nature and development of metacognition*, Hillside, NJ: Lawrence Erlbaum.
- Horn, R. E. (1998) *Visual language: Global communication for the 21st century*, MacroVU, Inc. Bainbridge Island, WA, 1998.
- Hyerle, D. (1995). *Thinking Maps: seeing is understanding*. Educational Leadership, vol. 53, no. 4, 1995, p. 85+. Business Economics and Theory.
- Hyerle, D. (1996). *Visual tools for constructing knowledge*. Alexandria, Virginia: Association for Supervision and Curriculum Development.

- Hyerle D. & Gray Matter Software (2007). *Thinking Maps* (Version 2.0, Innovative Learning Group) [Computer software]. Raleigh, NC; Innovative Sciences.
- Hyerle D. & L. Alper (2011). *Student successes with Thinking Maps®*. Second Edition. Thousand Oaks, California: Corwin.
- Inan, F. A., & Lowther, D. L. (2010). Factors affecting technology integration in K-12 classrooms: a path model. *Educational Technology Research and Development*, 58(2), 137–154.  
<https://doi.org/10.1007/s11423-009-9132-y>
- Lee, M. & Baylor, A. L. (2006). Designing metacognitive maps for web-based learning. *Educational Technology & Society*, 9(1). Retrieved from <http://cyberleninka.ru/article/n/designing-metacognitive-maps-for-web-based-learning>
- Lee, M. J., & Ko, A. J. (2015). Comparing the effectiveness of online learning approaches on CS1 learning outcomes, presented at eleventh annual International Conference on International Computing Education Research, Omaha, Nebraska (pp. 237–246). New York, NY: ACM Press.  
<https://doi.org/10.1145/2787622.2787709>
- Lin, X., Hmelo, C., Kinzer, C. K., & Secules, T. J. (1999). Designing technology to support reflection. *Educational Technology Research and Development*, 47(3), 43–62.
- Lin, X. (2001). Designing metacognitive activities. *Educational Technology Research and Development*, 49(2), 23–40.
- Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, problem solving, and self-awareness: Effects of explicit guidance, presented at ACM Conference on Human Factors in Computing, Santa Clara, California (pp. 1449–1461). New York, NY: ACM Press. <https://doi.org/10.1145/2858036.2858252>

- Mazumder, Q. H., & Finney, M. J. (2012). Improvement of confidence and motivation using online metacognition tool. *American Journal of Engineering Education*, 3(1), 53.
- Oliver, R., & Herrington, J. (1995). Developing effective hypermedia instructional materials. *Australasian Journal of Educational Technology*, 11(2), 8–22.
- Opfermann, M., Scheiter, K., Gerjets, P., & Schmeck A. Hypermedia and Self-Regulation: An interplay in both directions. In Azevedo, R., & Alevan, V. (Eds.). (2013). *International Handbook of Metacognition and Learning Technologies* (Vol. 28). New York, NY: Springer New York. <https://doi.org/10.1007/978-1-4419-5546-3>
- Robberecht, R. (2007). Interactive nonlinear learning environments. *Electronic Journal of E-Learning*, 5(1), 59–68.
- Shute, V. J. (2011). Stealth assessment in computer-based games to support learning. In S. Tobias & J. D. Fletcher (Eds.), *Computer Games and Instruction*, (pp. 503–524). Charlotte, NC: Information Age Publishers.
- Tufte, E. (1990). *Envisioning information*. Chesire: Connecticut.
- Tufte, E. (1997). *Visual explanations*. Chesire: Connecticut.
- Tufte, E. (2001). *The visual display of quantitative information*. Second Edition. Chesire: Connecticut.
- Tufte, E. (2006). *Beautiful evidence*. Chesire: Connecticut.
- Volet, S. E. (1991). Modelling and coaching of relevant metacognitive strategies for enhancing university students' learning. *Learning and Instruction*, 1(4), 319–336. [https://doi.org/10.1016/0959-4752\(91\)90012-W](https://doi.org/10.1016/0959-4752(91)90012-W)
- W3schools. (n.d.). Retrieved May 1, 2017 from [https://www.w3schools.com/js/js\\_syntax.asp](https://www.w3schools.com/js/js_syntax.asp)

Yeung, N. & Summerfield, C. (2012). Metacognition in human decision-making: Confidence and error monitoring. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1594), 1310–1321. <https://doi.org/10.1098/rstb.2011.0416>

# Appendix

CURRICULUM: SeeSe, programming map to see sequences of code in JavaScript

Understanding JavaScript variables using maps to represent sequences of values

SeeSe is a mapping tool for learning basic aspects of JavaScript variables in computer programming with a focus on teaching the *structural* aspects of programming by making visual thinking processes visible. The main objective of this tool is teaching students *how to think* when programming specifically by fostering their visual-spatial and metacognitive skills. It is grounded in the revised theory that generate maps to make thinking processes visible facilitate student reflection and orientation.

A SeeSe map consists of words and shapes integrated into a single representation. It is composed by a) shapes with the form of containers that store values written as words or numbers, and b) connectors that follow a set of rules to put the shapes together in order to make visible the sequential aspect of the code. Using these two components students can transate a code written in JavaScript using a SeeSe map. The resulting visual representation may reflect students' understanding of the sequence of statements and their evaluation of values through the code.

## Learners

-Adult students with no previous experience of coding in JavaScript

Assumptions (students' previous knowledge of JavaScript)

-Strings

-Numbers

-Operators: Basic arithmetic operators / String operators

## Materials

-Paper and pencil for students

-Whiteboard for teacher

-NO computers

## Overview

5 sessions of 2 hours each

- Session 1 = Unit 1 / Variables
- Session 2 = Unit 2 / Functions (In development)
- Session 3 = Unit 3 / Variables + Functions (In development)
- Session 4 = Quiz
- Session 5 = re-Quiz and discussion



## UNIT 1 / Variables

Outcome:

Understanding the concept of variable using the box analogy.  
Acquisition of the visual syntax for variables.

Knowledge (Students will understand that)

- Variables are like boxes
- Each variable box need a label or name that is arbitrary but preferable meaningful
- Variable boxes can (and cannot) store values and those values can be anything
- Possible values to store are: numbers, strings, other variables, combinations using operators
- The values stored can be changed in the same way that you change the content of a box
- Previous values stored disappear after you put another value inside

*Predictable misunderstanding to tackle:*

- What happens with the values that were previously stored in a variable

Skills (Students will be able to)

- Create variables using the box visualization
- Name variables with arbitrary names using the label visualization
- Store values inside variables of different kinds including other variables and operators
- Change values stored in the boxes
- Translate a code wrote in JavaScript to the visual language and vice versa.

Essential Questions

- Do you have any box in your room? What do you store inside? Have you ever used it previously to store other things? Do you put labels to identify your boxes?
- How would you name your variable box? Why?
- What would you store inside?
- If I give you something different, could you store it inside your box?
- What would happen to the things that you had stored inside previously?
- When do we need to define variables? Why variables are useful?
- Which is the value of x variable in this point of the code? Wich is in this other point?

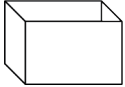
Assessment: Learning Evidence

- Peer review of their translation sheets, self-explanations, analysis of how students are using the visual language

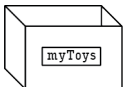
Learning Activities

1. Introduction to variables and values

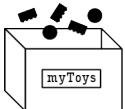
- Variables are like boxes



- You can put a label or name to your box which can be arbitrary but preferable meaningful. When you name a variable you are *defining* the variable. For example you can define a variable with the name of myToys



- Variable boxes can (and cannot) store things. Those things are known as the value of the variable. You can store anything. For example, you can store LEGOS into your myToys box



2. Changing values

- The value stored in the variable can be changed to new values. For example, if you grew up you will want to use the same box to store your new toys, like your iPod



- As a difference with real like, in the case of variables boxes previous values stored will disappear

3. Introducing the visual language

*To simplify our visual language, we are going to use the following visual symbols:*

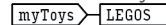
This is the icon for variable:



This is the icon for a variable with a name or label



This is how you represent the value stored in a variable



This is how you represent that a new value has been defined



\*Notice that we are using a sequence flow structure to represent how the values change in a variable. This structure is similar to a timeline because we need to represent successive stages of the same thing, in this case different values of a same variable.

4. Translate a code wrote in JavaScript syntax to the visual syntax and vice verse (next page)

Examples of translation

JavaScript Syntax	Visual Syntax
<pre>var age = 8 age = 17 age = 35</pre>	
<pre>var pizzaIngredients = "tomato, onion, mushroom" pizzaIngredients = "cheese"</pre>	
<pre>var secretNumber = 6 secretNumber = age secretNumber = age + 2</pre>	<p>(*Good practice: Notice that the length of the line after 6 is longer to match the current value of age variable on the top)</p>

Translate these JavaScript variables into the visual syntax

JavaScript Syntax	Visual Syntax
<pre>var temperatureC° = 15 temperatureC° = 30 temperatureC° = 13 temperatureC° = 16</pre>	
<pre>var sports = "football, tennis and yoga" sports = "pin-pong and gymnastics" sports = "golf and tennis"</pre>	
<pre>var colorRGB = "13, 55, 255" colorRGB = "33, 21, 12" colorRGB = "0, 0, 0"</pre>	

Translate to JavaScript syntax these visual variables

JavaScript Syntax	Visual Syntax

Create your own variables

JavaScript Syntax	Visual Syntax

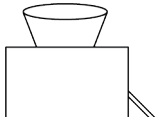






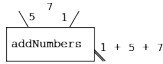
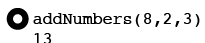
## UNIT 2 / Functions

\*NOTE: The graphic representation of functions is still under development. The new version will better represent how the order of the inputs affects the output, the role of functions as a sub-routine within the whole sequence of the code, and an integration of variable containers as parameters of the function and values as arguments.

Outcome:  
Understanding the concept of function using the machine analogy.  
Adquisition of the visual syntaxis for functions.

Knowledge (Students will understand that)	Skills (Students will be able to)
<ul style="list-style-type: none"> <li>- Functions are like machines. Machines do the things that we program them to do, but no other things.</li> <li>- Functions as machines have a name that could be arbitrary</li> <li>- Functions as well as machines can be installed, or defined</li> <li>- Functions as well as machines need an input or raw materials to do things. Those inputs are known as <i>arguments</i>.</li> <li>- For machines to do things you need to press the button. In functions, you have to <i>call</i> them.</li> </ul> <p><i>Predictable misunderstanding to tackle:</i> -Difference between defining a function and calling a function: When you are defining a function it is not doing anything yet, in the same way that an ATM can be installed but not doing anything until you call him by inserting your card</p>	<ul style="list-style-type: none"> <li>- Create functions using the machine visualization</li> <li>- Being able to see how using different arguments in a same function generate different returns</li> <li>- Translate functions wrote in JavaScript to the visual language and vice versa.</li> </ul>

Essential Questions	Assessment: Learning Evidence
<ul style="list-style-type: none"> <li>-What are examples of machines that you commonly use?</li> <li>-What does a coffee machine do and what it not do?</li> <li>-What machines need to have in order to work? (e.g.: ATM)</li> </ul>	<ul style="list-style-type: none"> <li>- Peer review of their translation sheets, self-explanations, analysis of how students are using the visual language</li> </ul>

Learning Activities	
<p>1.Introduction to functions -Functions are like machines. They do things that we program them to do, but no other things. For example, we want to do a coffee machine that if we put inside it coffee beans, water and sugar it will give us a cup of coffee.</p> <p>-To install the machine the <b>first</b> step is give it a name, it could be an arbitrary name but preferable meaningful.</p> <p>-The <b>second</b> step is to decide what the machine needs in order to do something. In this case the machine needs some beans, something liquid and sweetener. These are known as <i>paramenters</i>.</p> <p>-The <b>third</b> step is to decide what the machine is going to do with those arguments. In this case, it will be a cup of coffee. This is what the function <i>returns</i>.The machine is already installed it can be said that it has been <i>defined</i></p> <p>-You can install a machine as we have previously done but if you never feed it with raw materials it will never do anything. We are going to put coffee beans, water and sugar into this machine. These inputs are going to be the <i>arguments</i>.</p> <p>-Lastly, if you don't press the button of the machine it will never do anything. Pressing the button of a function is known as <i>calling</i> the function. We can call a function by saying its name and specifying the arguments For example, we call the coffeemaker with coffee beans, water and sugar and it will return us a cup of coffee.</p>	<div style="display: flex; flex-direction: column; align-items: center;">       </div> <p>2. Introducing the visual language - <i>To simplify our visual language, we are going to use the following primitives:</i></p> <ul style="list-style-type: none"> <li>- This is the icon for a function</li> </ul>  <ul style="list-style-type: none"> <li>- This is the icon to define a function, with parameters, name and what it is going to return</li> </ul>  <ul style="list-style-type: none"> <li>-This is how we represent that we are calling a function that have been previously defined</li> </ul>  <p>3.Translate a code wrote in JavaScript syntax to the visual language (next page)</p>

Examples of translation

JavaScript Syntax	Visual Syntax
<pre>function convertToF(temperatureC)   return temperatureC * 1.8 + 32</pre>	
<pre>convertToF(25)</pre>	<ul style="list-style-type: none"> <li>● convertToF(25) 77</li> </ul>
<pre>function makeLunch(food, moreFood)   return food + " and " + moreFood</pre>	
<pre>makeLunch (chips, fish)   return chips + " and " + fish</pre>	<ul style="list-style-type: none"> <li>● makeLunch(chips, fish) chips and fish</li> </ul> <p>(*Notice that the order of the arguments is important and as a result we get for lunch chips and fish and not fish and chips</p>

Translate this JavaScript function to the visual syntax

JavaScript Syntax	Visual Syntax
<pre>function getSquareArea(side)   return side * side</pre>	
<pre>getSquareArea(8)</pre>	
<pre>function yourFuture(numberOfChildrens, TypeOfHouse)   return "You will have " + numberOfChildrens + "and live in a "   + typeOfHouse</pre>	
<pre>yourFuture(20, Mansion)</pre>	

Make your own functions and write them in JavaScript Syntax and visual Syntax

JavaScript Syntax	Visual Syntax

## UNIT 3 / Variables and Functions

**Outcome:**

Understand how variables and functions can work together and how this can be represented visually by using certain structure and rules to organize them (grammar of the visual language)

Knowledge (Students will understand that)	Skills (Students will be able to)
<ul style="list-style-type: none"> <li>- Functions can use values stored in variables</li> <li>- Variables can store functions</li> <li>- Functions can be defined to change the values of variables</li> </ul>	<ul style="list-style-type: none"> <li>- Create functions using pre-defined values stored in variables</li> <li>- Create functions defined to change the values of variables</li> <li>- Translate Javascript code composed by interrelated functions and variables to the visual language using visual syntax and visual structure or grammar.</li> </ul>
Essential Questions	Assessment: Learning Evidence
<ul style="list-style-type: none"> <li>-What is the purpose of combining functions and variables?</li> <li>-Could the value of a variable be a function?</li> <li>-Which is the value of x variable in this point of the code? Wich is in this other point?</li> <li>-What specific differences you notice between your visual map and the one of your classmate?</li> </ul>	<ol style="list-style-type: none"> <li>1. Quiz with complex set of interrelated variables and functions that should be solved using the visual language</li> <li>2. Correction of the quiz</li> <li>3. Peers work to generate a new visual representation that solve the quiz correctly</li> <li>4. Second quiz</li> <li>5. Correction of the quiz and analysis of progress in terms of results and quality of their visual representations</li> </ol>

**Learning Activities**

1. Learning visual structure and rules to organize our already acquired visual vocabulary:

- In Class exercise (next page). Each student will receive a JavaScript code and a large sheet of paper to make the translation to the visual language following these instructions that will guide them through good practices:

- a. Rotate the sheet of paper to use it in a horizontal way
- b. Draw a horizontal line approx. 3 inches below the top of the page
- c. You will use the top side of the page to “install” your machine functions and the bottom side to draw your variables boxes
- d. Read the code line by line and for each line draw the correspondent representation. If it is a function draw in the top, if it is a variable draw it below the line.
- e. When it is a variable first draw the variable label on the left and then it’s value on a box connected by a line.
- f. Draw the next variable below the previous one and connect it’s value with a line that is longer that the previous one, so that you can take in consideration the values of previous variables when making this new one in the case that you need them.

\*Note: all these instructions seems complicated but they are pretty simple when you put them in practice, so is important that the teacher makes a clear demonstratration of the process.

Make a visual representation of this JavaScript code using the visual language

```
function getReady(jump, sing);
    return jump + " and + " sing;

function makeExercise(bike, helmet);
    return "pick your " + getReady(bike, helmet) + " and go";

function getDress(sneakers);
    return sneakers;

function getStrong(mix, swim, sing);
    return mix(swim, sing);

function askSiri(what)
    return "It's a perfect day for " + what

var clothes = "shorts";

var extraClothes = "gloves";

extraClothes = clothes + extraClothes

clothes = "pants";

clothes = getReady(extraClothes, clothes);

var running = makeExercise(clothes, extraClothes);

var record = getReady;

var moreExercises = getDress(record);

var jumping = moreExercises("jump", "clap");

var last = getStrong(record, "sing", "walk");

var justDoSomething = askSiri(last);
```

**Question: What is the value of justDoSomething?**

(Answer in the next page)

