Permeable media: A design strategy for Constructionist software

Permeable media

Chris Proctor[1], Yeshi Paljor[1], and Varun Bhatt[1]

[1] University at Buffalo (SUNY), Buffalo, New York, USA

Abstract

This paper introduces a design strategy called permeable media for software used in Constructionist approaches to introductory computer science education. Permeable media is characterized by three qualities: it invites learners to extend themselves into the medium, it has affordances for learners to make the medium part of themselves, and it supports learners in growing beyond the medium when they are ready to do so. This paper joins a long tradition of design for Constructionist learning environments, emphasizing two principles which have not been emphasized in the prior literature: incorporating media into one's identity and embodiment, and support for growing beyond the medium. This paper illustrates permeable media by analyzing the design of Banjo, a software package which allows beginners to create web applications. The final sections theorize the relationship between permeable media and computational literacies and propose a research agenda based on this paper's conceptualization of permeable media.

Keywords and Phrases: Permeable media, User interface design, Computer science education, Computational literacies

1. Introduction

This paper introduces a design strategy called *permeable media* for software used in Constructionist approaches to introductory computer science. We define permeable media in terms of three qualities: (1) permeable media invites beginners to extend themselves into the medium, (2) permeable media offers affordances for integrating it into learners' cognition and identities, and (3) permeable media supports learners to keep growing beyond the media when they are ready. The metaphor we have in mind is potting soil: a medium in which seedlings can germinate, integrate the soil into their root systems, and then continue to expand into the surrounding soil after they are transplanted.

This paper develops a theoretical account of permeable media, drawing on theories of distributed cognition, embodiment, identity authorship, and material intelligence. We then analyze a case study of Banjo, a software package designed as permeable

Proctor, C., Paljor, Y., & Bhatt, V. (2025). Permeable media: A design strategy for Constructionist software. Permeable media. *Constructionism Conference Proceedings, 8/2025*, 239–248. https://doi.org/10.21240/constr/2025/51.X

media, in the context of a Constructionist introductory computer science course (Proctor et al., 2020). We show the alignment of Banjo's permeable design strategy with the broader goals of the course.

2. Background

We view permeable media as following in the tradition of Resnick and Silverman's (2005) insights on designing construction kits for children, especially their strategies for inviting in diverse learners such as "low floor and wide walls" and "support many paths, many styles." Along with Resnick and Silverman, and many others, our aim is to design technologies well-suited to Constructionist learning (Papert, 1980). However, there are two aspects of permeable media which have not been emphasized in the prior literature: incorporating media into oneself and growing beyond the media.

Permeable media has affordances by which users can extend their cognitive and social practices into the medium, using it as an object-to-think-with (Papert 1980) and as a form of distributed cognition (Hollan, Hutchins, & Kirsh, 2000). The medium becomes part of the user in two senses. First, we mean that the user comes to experience the medium as part of their embodiment (Hirose, 2002), as an extension of their capacity to sense and to act on their environment. Cell phones are an excellent example of this kind of embodiment; one often has an awareness of where their phone is and may experience a moment of panic upon realizing it is missing. We tend to regard the phone's data and apps as extensions of our inner worlds and may experience a sense of violation if someone gains access to our phone without permission. We also act on the world through our phones, reaching for a phone during a conversation to look up information for a claim, to check the weather, or to find out how long we will have to wait for the bus. Second, we mean that users author identities for themselves (Holland et al., 1998; Ivanič, 1998) using the medium, such that the medium becomes inextricably woven into their self-conceptions and the social practices by which they sustain identities. For example, Brock Jr. (2020) documents distinctive forms of identity which only emerge within the figured world of Black Twitter, and which were dependent on the technological infrastructure of Twitter. One sense, therefore, in which media can be permeable is that users can extend themselves into it make it part of themselves.

The second quality which distinguishes permeable media from prior theorizations of Constructionist learning technologies is that users, when they are ready, can continue growing beyond the affordances and constraints of the media without having to abandon the practices and understandings they have built. This has been a challenge for Scratch, for example, as learners often perceive Scratch to be juvenile and express a desire to move on to "real programming" (Weintrop & Wilensky, 2015a). However, computer science concepts appear to be bound to some extent to the programming modality (Parsons & Haden, 2007; Weintrop & Wilensky, 2015b), so transfer to a language such as Python may be difficult and incomplete. A more permeable approach might be to provide an interface which can translate between block-based and text-based representations, or to work from the beginning in a full-powered programming modality, with layers of abstraction to help beginners manage complexity. For example, Racket (Felleisen et al., 2015) has been used as a teaching language in which the language itself is initially very limited and then is gradually extended as new constructs are learned. Such strategies could make it possible to keep growing beyond the novice stage without having to leave behind the media into which one has invested oneself.

Permeable media stands in contrast to several existing framings of the role of technology and learning. When technology is viewed as scaffolding (Pea, 2004), it is seen as a temporary external support which will later be removed so that the learner can perform the skill on their own. In contrast, permeable media becomes part of the learner (as described above), and the learner's newly acquired competencies are often dependent on their newly configured chimeric embodiment. Permeable media also stands in strong contrast with the sandbox model of teaching computer science, which aims for conceptual learning through decontextualized exercises, such that the student never has the freedom to use the medium in unprescribed ways and is never in the position to decide what to do with the medium.

3. Context

Before analyzing Banjo (a software package designed as permeable media) we briefly introduce the course for which it was designed, to show how the design of the software functions within the broader learning ecology. Making With Code (Proctor et al., 2020) is a Constructionist (Papert, 1980) introductory computer science course for early high school-aged students. The course learning objectives are framed in terms of Kafai and Proctor's (2021) computational literacies framework, considering cognitive, situated, and critical framings of computational thinking (See Figure 1). These framings are distinguished in terms of scale as well as epistemology. The cognitive framing views learning as primarily a matter of individual skills and knowledge; Making With Code's cognitive goals are expressed as forming personal relationships with powerful ideas, in the tradition of Papert. The situated framing recognizes the importance of cognitive processes but views the learner's identity and the meaning of her actions to be produced through ongoing participation in a community of practice. Making With Code's situated goals are focused on cultivating "the computer cultures that may develop everywhere in the next decades" (Papert, 1980, p. 20). Finally, the critical framing zooms out even further, considering individuals and communities of practice within the context of broader cultural discourses and power structures. Making With Code's critical learning goals are conceptualized as a computational extension of Freire and Macedo's "reading the word and reading the world" (1987), applying their formal computer science learning to better understand the technological infrastructure which shapes their identities and social interactions.

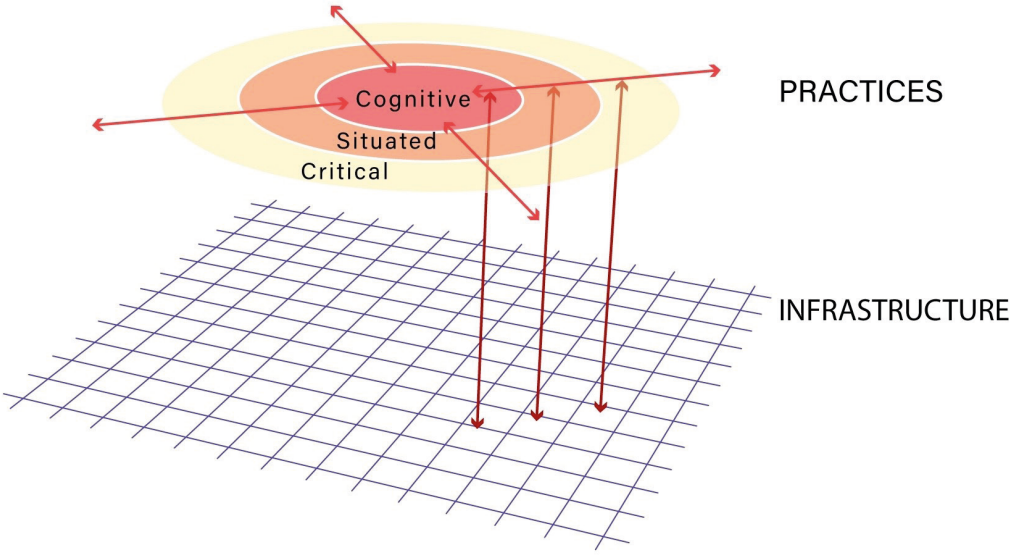

Figure 1: Diagrammatic representation of computational literacies.

Figure 1 depicts cognitive, situated, and critical learning as concentric circles within a plane of practice, hovering above a plane of infrastructure. The vertical axis connecting practices to infrastructure reflects the recognition that many domains of our lives are now mediated by computational platforms which shape and surveil us (Zuboff, 2019), as well as the primary pedagogical approach of the course: working closely with powerful tools. Recognizing that individual cognition, social interaction, and critical consciousness today involve interaction with computational media, Making With Code prioritizes students learning to use tools such as the command line, a code editor, and software documentation, guided by Papert's (1980) concept of objects-to-think-with and diSessa's (2001) concept of material intelligence. Each unit in the course consists of a series of labs which introduce new ideas and invite students to explore them via custom software packages, and then each unit concludes with a project in which students apply the new ideas in personally-meaningful projects. The design strategy of permeable media supports the materialization of ideas in media, thereby playing a central role in the broader pedagogical approach of the course.

4. Banjo: A case study in the design of permeable Media

Banjo (Proctor, 2025) is a Python software package which provides an abstraction layer over Django, a widely-used web application framework. Despite Django's excellent documentation, it is not suitable for first-year students because there are simply too many new ideas at play in a web application. Within the Making with Code curriculum, Banjo is introduced at the beginning of a unit focused on networking and system design, in a lab specifically focused on the design of client/server distributed systems, and the structure and content of data sent between clients and servers. One lab considers how a client program makes HTTP requests and works with responses; the following lab uses Banjo to explore how a server receives HTTP requests and generates responses using the model/view design pattern. Banjo was designed as permeable media to explore these ideas: welcoming to beginners, supporting users in making the medium part of themselves, and allowing users to grow through and beyond it (in this case, into working with Django).

4.1 Invite users into the medium

Several features of Banjo's design help make it appealing to beginner users. First, Banjo aims to minimize cognitive load by narrowly scoping the problem domain and exposing a simplified interface to the user. A Banjo server only accepts HTTP requests at statically declared routes with specified parameters, and only responds with JSON data. Banjo does not support HTML responses (e.g. web pages), to help students focus on creating algorithmically-interesting programs rather than getting lost learning the endless details of HTML and CSS. The labs and projects within which Banjo is introduced encourage student curiosity and creativity in exploring what they can do with a tool so simple they can understand it fully.

Web applications typically consist of many files; navigating between them to write new features or debug errors can be a major source of cognitive load for beginners. A Banjo app, in contrast, consists of exactly two files: models.py and views.py. (The code for a small but complete example app, which hosts riddles and allows other users to guess their answers, is available at https://git.makingwithcode.org/archive/banjo-demo.) Additionally, configuring and running a web application is often complex; Banjo can be installed from the Python Package Index and then run with the command `banjo`. The simplicity and approachability of Banjo is reinforced with

beginner-friendly documentation, supporting an important CS reading practice and making it more likely that beginners will have productive experiences seeking answers to problems.

Banjo makes heavy use of selective disclosure (Hmelo & Guzdial, 1996; Resnick, Berg, & Eisenberg, 2000), intentionally exposing some aspects of the underlying complexity and hiding others. For example, users declare a database schema using Django's object-relational mapping, but a severely limited subset of model field types is provided: booleans, strings, integers, floats, and foreign keys. Sensible defaults are selected for each model field, and most of the options supported by Django are hidden. The result is a powerful but narrowly-scoped introduction to databases as a persistent data store behind a web application.

Selective disclosure is also used to present a simplified conceptual model of the request/response lifecycle. In Banjo, each request/response is modeled as a function which receives a single dict as input and which also returns a single dict. (Students will have already been introduced to functional programming as a problem-solving paradigm, to lists and dicts as data structures, and will have experience navigating composite data structures as they parsed a server's JSON response.) Several exceptions are provided which model HTTP client errors, such as Forbidden (403), Not Found (404), and Not Allowed (405). By modeling the complex and unfamiliar behavior of a web application as a straightforward function whose job is to transform a request into a response, Banjo supports learners in applying a simple but powerful abstraction to web processes which they probably use every day but may not have felt they could comprehend.

Finally, Banjo invites beginners by making its behavior visible so that users can see the result of changes and can debug errors. Debugging a client/server application can be more complex than the simple programs students will have worked with previously, so it is important that beginners can see the running application from the perspective of the server (the terminal session logs requests handled, as well as any messages printed by the user) and from the perspective of the client (the browser uses Django's debug mode to display details when something goes wrong). The app's routes, declared using function decorators in views.py (a strategy borrowed from Flask), require request parameters to be specified; type-checking is handled automatically. This allows students to clearly see the input data to their view functions and avoids subtle errors which can arise with unexpected input. API views, shown in Figure 2, are automatically generated for each route, allowing beginners to test out their applications and to show them off to others.

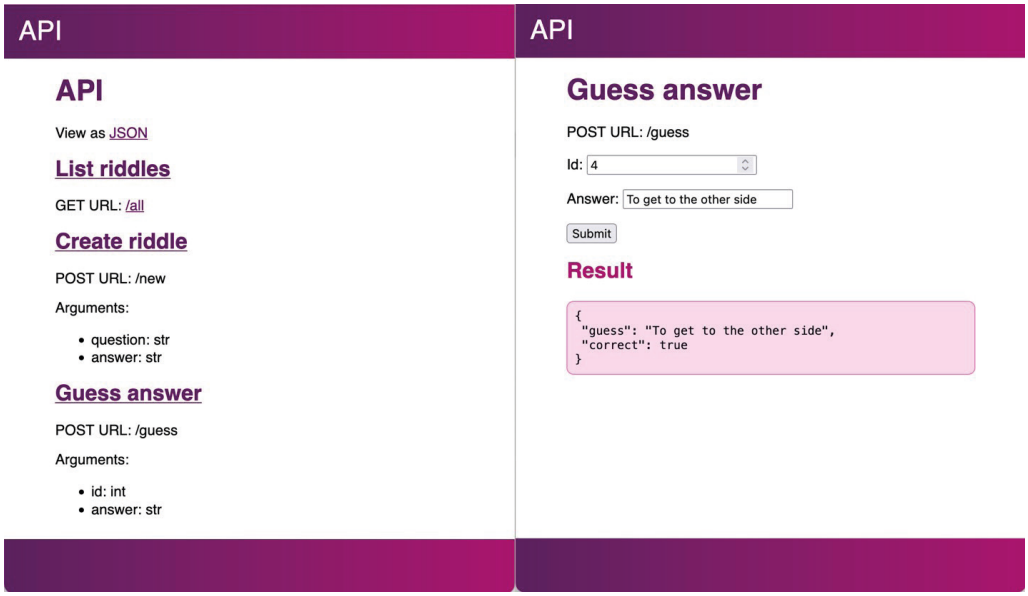

Figure 2: Screenshots of Banjo's built-in API views. The left screenshot shows a listing of all defined routes; the right screenshot shows an automatically generated form for a route accepting POST requests.

4.2 Supports users making the medium part of themselves

Banjo was designed with affordances which encourage users to extend their cognitive and social practices into Banjo (incorporating it into their identity), and to sense and act on the world through the medium (incorporating it into their embodiment). While the affordances described in this section are part of Banjo, they become more salient to users within the pedagogical context in which Banjo is used.

Banjo supports learners incorporating it into their cognition through its minimal interface, foregrounding computational structures which can serve as powerful objects-to-think-with (Papert, 1980). In the models file, database tables are declared as classes and table columns are declared as class attributes. This object-relational mapping, borrowed from Django, is a very powerful abstraction for modeling domain problems, presented by Banjo in an approachable way. The views file, meanwhile, implements server request/response cycles as functions, making it easy for learners to reason about what kinds of data are being sent in requests, what kind of processing the server is doing, and what kinds of data are being returned in responses. During runtime, users can interact with the server as clients, making GET and POST requests and viewing the responses, and can also see the server's log of handled requests. Later units in Making With Code take advantage of these affordances, using Banjo as a cognitive tool. For example, a later lab on public key cryptography uses Banjo to build a simple encrypted chat app. As students work out how to send and receive encrypted messages, they rely on their understanding of Banjo to reason about the security of data in transit and data stored in the app's database.

In addition to supporting cognition, Banjo supports learners incorporating it into social practices by making it easy to launch a working server with which others can interact. A single command, banjo, handles all the intermediate steps needed to update and launch a local server, including generating and applying database migrations. In Making With Code, the networking unit concludes with a unit project in which students design, build, and share a server written with Banjo. This makes available the possibility for a student project to become significant, not just for the teacher or

for a grade, but within the social world of the class or the school. In one school, for example, student Banjo projects are hosted on a machine accessible from the school's internal network, so that they can become part of the school's computational infrastructure, for playful purposes or even to solve real issues for the school, especially when students realize that they can apply the rest of their CS learning within a view function. Ethical issues regarding privacy, anonymity, and content moderation are then figured as serious, real-world issues rather than a curricular afterthought.

Imagining, designing, and building servers which might become shared social infrastructure encourages students to analyze other infrastructure which they already use and likely take for granted. For example, Wolf et al. (2023) document how, after studying how data flows through networks with Banjo, students describe "re-seeing" the world around them, and "growing as a person" as they become more aware of how intertwined many aspects of their lives are with computational technologies. Developing critical consciousness of how we are shaped by societal infrastructure is an explicit goal of Making With Code and is supported by reflection questions which become the basis of classroom discussion. One lab in which students work with Banjo concludes by asking students to connect their new understanding to their everyday lives: "Choose a program (Steam), web app (Google Docs), or app (Weather) that you use frequently. You can't observe the calls this program is making to its server (unless you have fancy tools), but you can infer some of the calls based on the program's behavior. Describe a few routes which you think may exist for your chosen program's backend server." Although this question does not specifically elicit reflections about power, it sets the stage for a classroom discussion which does.

4.3 Support users growing past the medium

Selective disclosure has sometimes been referred to as "glass-boxing" or "black-boxing," which highlights its binary nature: aspects of the underlying system are either disclosed or not. Permeable media employs selective disclosure to support beginners as discussed above, but the goal is to make the abstraction layer soft and permeable so that students can grow through the abstraction into underlying systems when they are ready to do so. One example of Banjo's permeable design is that all the model classes in Banjo are subclasses of Django classes, as described above. This means that if students want to work with data types not supported by Banjo, such as dates, they can import the relevant class from Django, as long as they are ready to engage with Django's more complex features and documentation, and to engage with the fundamental complexity of representing dates and times. Although many components can be imported from the underlying framework, Banjo puts some fundamental limits on what can be built with its toolkit in the interests of the goals above. These constraints can themselves provide a learning opportunity, as more advanced students start to become aware of the tradeoffs built into Banjo's design and consider whether they are ready to work directly in Django.

Even when learners move on from Banjo, they do not need to leave behind the practices and mental models they developed, because the practices and models remain fundamentally the same in Django (and most other web application frameworks) even as they grow in complexity. The model/view server architecture remains the same, even though both models and views become much more capable and new layers of abstraction are introduced (e.g. middleware, more explicit interaction with the database). Banjo's file layout, two files within a directory, ends up being the minimal case for a Django project. The interfaces for running, testing, and debugging the

code remain are also largely the same. Banjo's permeable abstraction layer provides the simplicity beginners need while building conceptual models and practices which learners can keep using even as they grow beyond the medium. This is a significant contrast with beginner-friendly designs such as block-based programming which can be richly generative contexts for creating projects and exploring ideas, but which support practices for creating, testing, and debugging code which are quite distinct from text-based languages students might move on to next.

5. Discussion

The design strategy of permeable media emerged from our iterative efforts at building software to support the goals of Making With Code, theorized in terms of computational literacies. We hoped students would develop relationships with powerful ideas, participate in computer cultures, and gain critical consciousness of the ways their existing identities, relationships, and cultures are shaped by computing – and how they might deploy their formal CS learning to become morefree. These goals are represented in Figure 1 by the concentric circles of cognitive, situated, and critical practices. At the same time, we wanted to support students' learning to think with powerful tools, their material intelligence to use diSessa's (2001) term. This goal is represented by the vertical axis in Figure 1, connecting practices to infrastructure. After quite a bit of trial and error, we have produced a number of carefully-crafted software packages which work well in practice supporting the labs and projects in Making With Code. Permeable media is our theorization of what these designs have in common and why they work.

The primary mechanism by which permeable media supports these goals is by materializing students' practices: their individual cognition, social practices, and critical consciousness are realized through and with computational media. As we argue above, permeable media has the potential to be more than just a tool to be used when needed and then put down; it could become part of its users. This could be conceptualized either in terms of embodiment (emphasizing that our corporeal extensions into media are as concrete as flesh) or in terms of identity authorship (emphasizing that our identities are as semiotic as media). In either conceptualization, the stakes are raised as students' learning about computer science also means learning about themselves.

Thinking again about Figure 1, this materialization of learning would seem to relate most directly to the vertical axis, bringing practice closer to infrastructure. However, if students incorporate classroom learning as part of themselves, the infrastructure can also serve as a conduit between the scales and epistemologies of the radial axis traversing cognitive, situated, and critical practices. If students experience permeable media as part of themselves, then classroom learning (e.g. about how data moves between clients and servers) might also transform students' understandings of that same embodiment or identity in other contexts: while playing online games, having emotional conversations over text message, arguing with family members about whether claims from social media are true, or wondering whether democracy can survive in the digital age. Permeable media, then, may play an important role in computer science education theorized in terms of computational literacies.

6. Next Steps

In addition to being a design strategy, permeable media has potential as a conceptual framework for research. Working with Sandoval's (2014) conjecture mapping framework, we are currently using this paper's articulation of permeable media as a high-level conjecture and beginning to formally study its design conjectures (whether permeable media supports learners into the medium, making it part of themselves, and growing through it) and its theoretical conjectures (whether the practices supported by permeable media actually lead to cognitive, situated, and critical computer science learning).

Acknowledgement

This material is based upon work supported by the National Science Foundation under Award No. 2219433.

References

Brock Jr., André. 2020. *Distributed Blackness*. New York University Press.

diSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. MIT Press.

Felleisen, Matthias, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy, and Sam Tobin-Hochstadt. 2015. "The Racket Manifesto." In *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Freire, P., & Macedo, D. (1987). *Literacy: Reading the word and the world*. Routledge.

Hirose, N. (2002). An ecological approach to embodiment and cognition. *Cognitive Systems Research*, 3(3), 289–299. https://doi.org/10.1016/S1389-0417(02)00044-X

Hmelo, C. E., & Guzdial, M. (1996). Of black and glass boxes: Scaffolding for doing and learning. *Proceedings of the 1996 International Conference on Learning Sciences*, 128–134.

Hollan, James, Edwin Hutchins, and David Kirsh. 2000. "Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research." *ACM Transactions on Computer-Human Interaction* 7 (2): 174–96. https://doi.org/10.1145/353485.353487.

Holland, Dorothy C, William Lachicotte Jr, Debra Skinner, and Carole Cain. 1998. *Identity and Agency in Cultural Worlds*. Harvard University Press.

Ivanič, R. (1998). *Writing and identity: The discoursal construction of identity in academic contexts*. Amsterdam: John Benjamins Publishing Co.

Kafai, Yasmin B, and Chris Proctor. 2021. "A Revaluation of Computational Thinking in K-12 Education: Moving Towards Computational Literacies." *Educational Researcher* 51 (2): 146–51. https://doi.org/10.3102/0013189X211057904.

Papert, Seymour. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc.

Parsons, Dale, and Patricia Haden. 2007. "Programming Osmosis: Knowledge Transfer from Imperative to Visual Programming Environments." In *Procedings of the Twentieth Annual NACCQ Conference*, 209–15. Citeseer.

Pea, Roy D. 2004. "The Social and Technological Dimensions of Scaffolding and Related Theoretical Concepts for Learning, Education, and Human Activity." *Journal of the Learning Sciences* 13 (3): 423–51.

Proctor, C. (2025). Banjo (Version 0.9.1). [Computer Software]. Retrieved from https://github.com/cproctor/django-banjo

Proctor, C., Han, J., Wolf, J., Ng, K., & Blikstein, P. (2020). Recovering Constructionism in computer science: Design of a ninth-grade introductory computer science course. In B. Tangney, J. Rowan Byrne, & C. Girvan (Eds.) Proceedings of the 2020 Constructionism Conference. (pp. 473-481). Dublin, Ireland: University of Dublin.

Resnick, M., Berg, R., & Eisenberg, M. (2000). Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation. *Journal of the Learning Sciences*, 9(1), 7–30. https://doi.org/10.1207/s15327809jls0901_3

Resnick, Mitchel, and Brian Silverman. 2005. "Some Reflections on Designing Construction Kits for Kids." In *Proceedings of the 2005 Conference on Interaction Design and Children*, 117–22. ACM. https://doi.org/10.1145/1109540.1109556.

Sandoval, W. (2014). Conjecture Mapping: An Approach to Systematic Educational Design Research. *Journal of the Learning Sciences*, 23(1), 18–36. https://doi.org/10.1080/10508406.2013.778204

Weintrop, David, and Uri Wilensky. 2015a. "To Block or Not to Block, That Is the Question: Students' Perceptions of Blocks-Based Programming." In *Proceedings of the 14th international conference on interaction design and children*, 199–208. ACM. https://doi.org/10.1145/2771839.2771860.

Weintrop, David, and Uri Wilensky. 2015b. "Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs." In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, 101–10. ACM. https://doi.org/10.1145/2787622.2787721.

Wolf, J., Han, J., Proctor, C., Brown, E., Pang, J., & Blikstein, P. (2023). "Growing as a person": Developing Identity and Agency Across Formal CS Education and Everyday Computing Contexts. In Building knowledge and sustaining our community, Proceedings of the 16th International Conference on Computer-Supported Collaborative Learning – CSCL 2023. Montreal, Canada: International Society of the Learning Sciences.

Zuboff, S. (2019). *The age of surveillance capitalism: The fight for a human future at the new frontier of power.* Profile Books.