UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Modernizing Women's Learning in Software Development: A Study on Constructionist Pedagogy and Networked Support

Permalink

https://escholarship.org/uc/item/9w05t8p8

Author

Hegab, Dahlia

Publication Date

2015

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NonCommercial-NoDerivatives License, availalbe at https://creativecommons.org/licenses/by-nc-nd/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Modernizing Women's Learning in Software Development: A Study on Constructionist Pedagogy and Networked Support

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

In Information and Computer Sciences

By

Dahlia Hegab

Thesis Committee:
Professor, Judith Olson, Chair
Associate Professor, Don Patterson
Professor, Debra Richardson

2015

© 2015 Dahlia Hegab

DEDICATION

To my parents, who have supported me in ways I cannot explain.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Thank you to my advisor, Judy Olson, whose endless support and encouragement pushed me to make this research better. You helped me get through writers block and talked me through the tough parts. I could not have done this without you. Also a special thank you to my committee members, Don Patterson and Debra Richardson. Your flexibility and support made this happen. I am beyond grateful to the PhD students that helped, in this, and my defense, especially Tao Wang, Yiran Wang, Martin Shelton, and Andy Echenique. Your wisdom and advice are appreciated more than you know. Also thank you to my peers at Hana Lab and Patterson Lab Group.

I am also beyond grateful to those at Hackbright Academy that made this work possible. To Angie Chang, Christian Fernandez, and David Phillips, and the students who volunteered to be interviewed, you were gracious and amazingly hospitable in allowing me to understand how you were making meaningful changes in the software development industry. This work is really for you and the possibility of the futures you are creating.

Lastly, this work would not be possible without the support of AAUW Selected Professions Fellowship, the Google Anita Borg Fellowship, and UCI's COR Graduate Student Fellowship.

To all those mentioned, and some of those who I might have forgotten, a heartfelt thank you!

ABSTRACT OF THE THESIS

Modernizing Women's Learning in Software Development: A Study on Constructionist Pedagogy and Networked Support

By

Dahlia Hegab

Master of Science in Information and Computer Sciences

University of California, Irvine

Professor Judith S. Olson, Chair

We present the results of a study of the learning practices of adult women in a 10 week software engineering bootcamp in San Francisco. We explore the technical, social, and pedagogical constructionist practices resulting from student immersion in this organization. The results of our research reveal how cultivating a distributed and collocated learning process that incorporates communal support at the peer and instructional levels, while providing a network of alums and industry mentors to encourage and refine career prospects, can facilitate successful inclusion and transitioning of non-technical women into the software engineering field.

Chapter 1- Introduction and Research Questions

Currently, there is a significant need for more professionals in the software development industry. There are 1.4 million jobs available and only 400,000 computer science students set to graduate in the last year (Soper 2014). In addition, only 17.4 % of those computer science graduates are women (Zweben and Bizot 2013). Various studies have divulged reasons why women do not end up in computer science including: "lack of experience, lack of confidence, a misperception of the field, a misperception of the CS culture as hostile and or "geeky", and a lack of role-models and mentors" (Klawe 2013; AAUW 2000). A growing body of literature (Hartness 2011; Milam 2012, Boyer et al. 2014; Margolis and Fisher 2003), (Weaver and Prey 2013) has sought to address concerns regarding gender diversity in computing. Numerous initiatives to resolve gender disparities in computing and engineering disciplines have included:

- computing outreach programs for girl scouts (Bruckman et al. 2009),
- summer camp engineering programs for middle school girls (Webb and Rosson 2011) ,
- the emergence of pre-introductory computer science courses in colleges (CS 0) (Margolis and Fisher 2003; Klawe 2013),
- the creation of makerspaces as communal spaces to incubate engineering learning (Blikstein 2013),
- informal DIY learning groups/meetup groups targeting women,

- school run Fablabs (Blikstein 2013)
- post-collegiate software engineering bootcamps.

Post collegiate software engineering bootcamps, specifically, have become popular in recent years. Today, there are at least 60 software bootcamps in existence across the globe and the number is quickly rising (Kamenetz 2015). These bootcamps range in accommodating specific populations (e.g., one is Christian based, another recruits underrepresented minorities, while another is all female) (Kamenetz 2015). None of them promise a degree and they vary in time commitments (usually around 10-12 weeks) and cost (approximately $10,000-$15,000) (Kamenetz 2015). What they do offer, however, is an opportunity for populations of people who have an interest in computing, but little experience in it, to get the experience needed to obtain a software engineering position.

In our study, we focus on one post-collegiate, all female, software engineering bootcamp, Hackbright Academy. We should note, Hackbright does not actually call itself a bootcamp. On its webpage, it describes itself as a "leading engineering school for women" with "a mission to increase female representation in tech through education, mentorship, and community" (Hackbright Academy 2015). It offers a ten week course to teach women software engineering basics, Python, and several other programming technologies (HTML, Javascript. JSON, JQuery, etc.). Moreover, it assists graduates in obtaining a software engineering position by holding a career day that includes interviews with 20-25 of its partner companies (including Pinterest, Facebook, SurveyMonkey, etc.) (Hackbright

Academy 2015) . This ten week course is described as “an accelerated software engineering fellowship” (Hackbright Academy 2015), presumably to demonstrate the high caliber of the program and it graduates.

It should be noted Hackbright Academy is distinctive as a bootcamp because of its dual objective: it hopes to address concerns of both lack of supply *and* lack of gender diversity in the software engineering industry. More importantly, it aims to combine and augment existing perspectives in educational design by introducing new ways of engaging those female students. It provides a constructionist environment that is supportive emotionally and professionally through accessible teaching staff and an industry mentorship program.

In the following work, we focus on the pedagogical practices, the social and professional environment, and the mentorship program provided at Hackbright Academy. We also look at the intelligent design of its educational space, its students’ use of pair programming, its demonstration of computer supported cooperative learning in the space, and its radical collocation practices that enhance its educational environment. We also evaluate the reach of its social and professional practices beyond student graduation.

Research Questions

Our research study is guided by several questions, which are as follows:

1. How does Hackbright compare with traditional learning environments and other informal learning environments?

2. What can be learned from the design of the learning space + its process + the resources in this environment to inform best practices in engineering literacy (among women)?

3. What kind of women sign up for Hackbright Academy? Who is attracted to it?

4. Does the manner of instruction or immersion of "non-technical" adult women at Hackbright affect their interest in this subject? How?

5. For women who have completed Hackbright, do they obtain industry positions? How frequently? What helps them stay?

6. What can be learned from pedagogical practices in this informal learning environment to inform best practices in engineering literacy of adult females?

7. What might we be able to learn from this population of "non-technical" adults transitioning into software engineering that could be relevant, if not applicable, to other populations trying to learn this subject?

Chapter 2- What is Hackbright?

Background Information:

- Participants are selected by going through a series of interviews, with a specific focus on evaluating their ability to explain a skill they are an expert in (the more clear and simplified the explanation, the better)
- Cohort is all-female (teaching staff is both male and female)

Learning Environment:

- Has a culture of respect and trust for learning
- Provides an undertone of support, complete openness, and vulnerability during the course (e.g. no questions are dumb questions)
- Offers an informal, yet semi structured learning environment
- Provides lectures for initial 5 weeks of the program, but the lectures are free form:
    - There are no formalized lesson plans
    - Lectures vary depending on questions raised.
- Uses many methods and tools to simplify learning including:
    - Teacher sessions/lectures,
    - Collaborative pair programming lab sessions,
    - Interactions with industry mentors (online and in person)

    - Distributed and collocated community (distributed interaction occurs through email listserv and during mentorship activities)
- Presents fundamental programming challenges to prepare students for their own projects
- Promotes self-motivated learning through personally meaningful projects
- Encourages students to work on personal artifacts through software project sites like Github

Social/Professional Support:

- Has student-led cross-class reunions, parties, and other types of social and career development events
- Has anonymized sources of support to encourage pushing through challenging times (i.e. a motivational wall of post-its put up by students)
- Provides an email listserv where people share social and professional information (including open jobs at their company or invitations to study an emerging technical practice)
- Provides unrestricted access to the learning space for current students and alums during and after the program.
- Creates emergent female community that can be readily seen in Hackbright's "females in tech" events

Professional Development:

- Provides up to 3 mentors per student (mentors from local tech companies volunteer and provide continued engagement with students personalized projects)
- Utilizes industry mentorship that includes distributed constructionist activities (help through chat, emails, Skype)
- Engages with current and popular technological culture (including vocational technologies used in partner companies such as Flask, Javascript, Python, Github, etc.)
- Offers career training with a focus on salary negotiations and interview preparation
- Provides new graduates with a career day where they pitch their projects to a representative from a reputable partner company looking to hire female candidates, such as Pinterest or Facebook (20-25 companies are represented on career day)
- Has open house events for Hackbright students to showcase and demo their projects to the public

Chapter 3- Related Works on Constructionist Theories

Learning Theories:

To enrich our understanding of the learning model we observed at Hackbright Academy, and expand on best practices in educational design, we turn towards the educational theory literature. We identified several theoretical models that emulated the primary goals and values of Hackbright Academy.

3.1 First Wave Constructionist Scholars: Papert, Piaget, Vygotsky

Part of the core ethos of Hackbright Academy, using individualized projects (and artifacts created during collaborative pair programming exercises) to demonstrate knowledge gained throughout the course, aligns well with the principles and assumptions of constructionism. Principles of constructionism include (Papert 1980):

1) *"knowledge being built by the learner,*
2) *an emphasis on having learners engaging in artifact constructions that are external and shared,*
3) *teachers having roles as facilitators of students' active learning."*

The concept of knowledge being built by the learner comes from Piaget's constructivism. Although Piaget and Papert concur on the need for knowledge to be built by the learner,

Papert diverts from Piaget by putting less emphasis on the cognitive processes of learning, and instead focuses on

1) the learners *constructing a physical object* to represent their learning, and
2) the learners' *cultural surroundings* (i.e. teachers, or tools in the learning process, or the environment itself).

Papert references the importance of culture in learning by pointing out (Ackermann 2001):

> *"All builders need materials to build with. Where I am at variance with Piaget is in the role I attribute to the surrounding cultures as a source of these materials. In some cases the culture supplies them in abundance, thus facilitating constructive Piagetian learning. But in many cases where Piaget would explain slower development of a particular concept by its greater complexity or formality, I see the critical factor as the relative poverty of the culture in those materials that would make the concept simple and concrete. In yet other cases, the culture may provide materials, but block their use."* (Ackermann 2001)

Papert discusses important components of learning culture, which may include teachers, tools in the learning process, and the environment itself. He highlights how these components facilitate or impede the learning process. In this sense, he draws on Piaget's concept of constructivism, but also ties it in with Vygotsky's socio-cultural theory, which posits that learning is social process (Ackermann 2001). Vygotsky emphasizes social

interaction in the development of learning by saying: “individual development cannot be understood without reference to the social and cultural context within which its embedded…higher mental processes in the individual have their origin in social processes” (McLeod 2007). In other words, community plays a central role in the process of learning or in “making meaning” of concepts (McLeod 2007).

3.2 Constructionism and Experiential Learning (Dewey)

Dewey posits similar sentiments to Papert, and Vygotsky, citing the importance that cultural components like teachers and learning environment can play, but is much more focused on itemizing what components are relevant factors in learning, and how they can affect it in either a positive or negative way. For example, Dewey states that education can be stifling to learners, and their independence, when they are taught that knowledge is transmitted in one direction, from the expert to the learner (Dewey 2007). He is one step removed from Papert, acknowledging that learners should transmit knowledge, but does explicitly state learning is their responsibility. Despite the difference in terminology for these scholars, it seems for the most part, both are stating the same thing: teachers play a part in facilitating learning, and, learners have to diagnose and resolve their own challenges as part of the learning process.

Dewey goes a step further in discussing potential concerns from the role that teachers play in a traditional classrooms, including the concern of keeping the order, instead of creating a

progressive environment where students are part of a community (Dewey 2007). This concept touches on concerns posited by later scholars, Freire and Blikstein, who worry that teachers are less concerned with learners' needs and more focused on maximizing their needs being met as instructors (Blikstein 2013). More importantly, Dewey and Papert both agree with earlier scholars, Vygotsky and Piaget, in stating that education and learning are social and interactive processes (Dewey 2007; Ackermann 2001). Through a review of scholars like Dewey and Papert, we see the origins of community based learning and later movements by scholars like Bruckman, Resnick, Blikstein, and Freire, who we will discuss later.

Dewey also cites the difficulties associated with traditional schools that are "insular" and therefore prevent real life interactions with the world (Dewey 2007). The effect is a lack of context for how the material learned fits into the world at large. Dewey is unique in citing the importance of this kind of learning, which he terms, "experiential learning" (Dewey 2007).

To clarify, the term "experiential learning," refers to the concept that students learning new material must find a way to ground unfamiliar concepts and ideas within the scope of ordinary life experience (Dewey 2007). Dewey also believes students' diverse backgrounds can create an infinitely diverse range of experiences for the educator to consider (Dewey 2007). Dewey also notes that it is the responsibility of teachers to organize learning experiences for a diverse range of students to be able to understand and engage with the

material. Developing this structure first requires acknowledgment of experience as a vehicle of learning. Subsequently the educator's discretion is important in selecting the material for a course of study and sensitivity to weaving connections between the students' previous experiences and new material, so that the value of lessons learned is maximized. One of Dewey's preeminent concerns was the educator's role in creating an environment that provided continuity and a contextualized model of student learning. The difficulty in this challenge lies in continually adapting subject matter as students' experiences grow and progress.

More importantly, Dewey (Dewey 2007), like Freire (Blikstein 2013), Margolis (Margolis and Fisher 2003), and Klawe (Klawe 2013), all touch on the importance of diversified perspectives in learning. Freire (Blikstein 2013), a critical pedagogy scholar, disapproves of de-contextualization of curriculum. In other words, he like Dewey, believe that learning needs to be grounded or "contextualized" in real world application. Papert (Ackermann 2001) echoes similar sentiments by criticizing uniformity in curriculum. He argues for diversity of ideas and experiences, like Dewey. Dewey (Dewey 2007) is very specific on the benefits of diversity of ideas, arguing that "democratic social arrangements," where all perspectives are considered and vocalized, promote a better quality of human experience that's more widely accessible and enjoyable than non-democratic ones. Freire, extends the concepts Papert and Dewey promote in valuing diversified perspectives, by introducing the idea of "culturally meaningful" curriculum construction, where designers get inspiration from the local culture toward creating "generative themes" with members of these cultures

(Blikstein 2013). Freire believes education is a tool of empowerment, and argues that "learners should go from the consciousness of the real to the consciousness of the possible as they perceive viable new alternatives beyond limiting situations" (Blikstein 2013). Therefore, he argues student projects should be deeply connected with meaningful problems, at a personal or community level. By doing this, he states, the design of student's solutions are both educational and empowering (Blikstein 2013).

Freire's concept of student projects being connected with meaningful problems in the world is echoed by current scholars and teachers including Margoilis (Margolis and Fisher 2003) and Klawe; Klawe 2013). Klawe introduces the importance of diversity in perspective in a different way than perhaps Dewey or Papert had imagined. In reconstructing the curriculum for Harvey Mudd's collegiate computing courses, Klawe replaced traditional "CS1" curriculum with a "breadth first approach" that provide students with substantial programming experience in a variety of application areas to some of the major intellectual and societal contributions on the field (Klawe 2013). Klawe, like Dewey and Freire, believes students thrive more in real-world experiences and research projects. She also advocates for students to engage in research to gain experience engaging in meaningful problem solving.

The second generation of constructionist thinkers, such as Papert's protégé, Resnick (Resnick and Rosenbaum 2013) and Amy Bruckman (Bruckman 1998), did not just speak of how to design generative learning environments, but created them (i.e. Scratch and

MediaMoo). Children, or new learners of computing, could solve meaningful problems, but the premise of their pedagogy was more focused on students to playing, learning, and making objects while learning. Resnick and Bruckman (i.e. MOOSE Crossing and Computer Clubhouse) were slightly different because they put more emphasis on creating a safe space for learners to engage with each other for the purpose of experimenting, sharing, and learning from each other. In contrast, experiential thinkers and critical pedagogists (i.e. Freire), focused more on having students create projects that were *contextually relevant,* instead focusing on a safe space or creating fun places to learn.

The new generation of making, which Freire could arguably be a part of, with scholars such as Blikstein, tries to merge *all* these learning pedagogies (Blikstein 2013). They try to create an experiential safe space that is not imagined or virtual, like some constructionist environments (i.e. MOOSE Crossing). Additionally, the new generation of making in constructionist environments provides *both* a safe space and a learning environment that is contextually applicable to the real world. This allows learners in engineering and computing disciplines to experiment and create relevant, innovative, and meaningful personal projects.

That is perhaps the introduction for our work, which discusses a safe space for learning engineering and computing practices that promotes diversity in learning and application of real world concerns. This application is demonstrated through a constructionist

environment predicated on vocational training for adult women entering the software engineering pipeline.

In our research, we focus on the emergence of this new constructionist environment, termed a "software engineering bootcamp". However, it must be noted, not all bootcamps are designed to be safe spaces where people can promoted diversified viewpoints while making things to learn vocational skills. The one we researched, Hackbright Academy, is specific in having these goals. Other bootcamps in this space are vastly different in approach and training. Some have been touted as "factories" where graduates taught material as quickly as possible in a competitive environment and then pumped out in the hopes of obtaining a new software developer position, but not necessarily assistance from the bootcamp in obtaining it.

As mentioned in Chapter 2, in our study of Hackbright Academy, an all-female software engineering bootcamp, we study the process and design of the learning space. We learn that is both supportive and constructionist. It employs experiential and critical pedagogical practices so students can create personally meaningful projects, while using their unique backgrounds and perspectives in doing so, *and* obtain employment positions (which require contextually relevant learning). Moreover, as female graduates become employed and transition in software engineering roles (particularly from non-technical fields), these students "create change" by contributing their thoughts and perspectives to the engineering and computing workforce. They, like subjects in Lindtner's recent work in

making at hackerspaces and hardware startups (Lindtner, Hertz, and Dourish 2014), use constructionist spaces not just as part of a hobbyist practice, but as a tool in professionalization.

3.3 Social Constructionism and Its Take on Culture in Education

Shaw, the protégé of Papert and Resnick, introduced the third wave of constructionism. Shaw emphasized the value in providing an adequate social setting for constructionist activities by discussing his study on urban neighborhood residents creating neighborhood programs to help each other (Shaw 1995). Using his MUSIC (Multi-User Sessions in the Community) software, Shaw created a digital network of neighborhood natives who built a total of 11 successfully organized and maintained projects. These included a group trip to Jamaica, a poetry collection, a summer jobs program for neighborhood teenagers, and crime watch information updates (Shaw 1995). What is important to note is his study's intent to enable members to invest in relationships in order to construct artifacts, which in this case, were neighborhood services and programs.

Shaw redefined constructionist artifacts to include local community organizations, written literary collections, and grassroots efforts. All of his constructionist activities were outside of traditional classrooms and did not involve common constructionist activities such as learning to code or building a robot. Instead, Shaw introduced constructionist activities that could transpire in the real world (grounding Papert's constructionism in Dewey's

experiential learning). Shaw also shed light on constructionist learning and social environments that could persist beyond one school year or one phase in a student's learning because they were rooted in organizations without those temporal considerations. He also took a step in informing scholars how to design constructionist environments that were not virtual, or based on constructionist kits, or simplified programming building blocks, but were instead grounded in real world environments.

Although Shaw's study focused on real world constructionism instead of virtual constructionism, his study did nonetheless provide several contributions to future virtual constructionist environments. His emphasis on the importance of social interactions in constructionism, is a precursor to much of the later research done by other constructionists like Amy Bruckman and Mitch Resnick in virtual learning environments (Shaw 1995; Bruckman 1998; Utting et al. 2010).

Bruckman extends Shaw's work in virtual learning environments such as MUDs (text based virtual reality environments), by explaining the value of social setting (i.e. social interactions) in these new learning environments. In Bruckman's MediaMoo and MOOSE Crossing (Bruckman 1998), she discovers that many beginners' motivation for trying to program something in MUDs are primarily social (Bruckman 1998). She notes that the first step in learning to program is the hardest and that the initial barrier is primarily emotional, hence a *community* provides the initial motivation for learning to program and provides support to help through the process (Bruckman 1998). Additionally, "an individual's quest

for mastery, if situated in social activity, leads other members in the community to act as an "appreciative audience" and the artifact created becomes a tool used for social contact and social status," that can contribute to society (Bruckman 1998). Hence, an appreciative audience for the artifacts helps to not only create one's identity in the community, it helps motivate learners to continually engage in the community (one student, Jim, in Bruckman's study said "while programming is fun, I don't think I'd do it if there wasn't anyone who would appreciate it") (Bruckman 1998).

Hence, it's important to note the social dynamics in constructionist activities become the catalyst for individuals to begin and continue their learning. Particularly if a learner is struggling, the value of the communal support provided by social environments becomes very evident. Constructed objects can follow from social activity in some instances, while in other instances they can become the predecessor to it. The value of the support received from social activities around learning is most evident to participants who do not receive it. Emotional and technical support (i.e. asking for help, receiving help, and giving help) are all social acts which help to build networks of relationships. Hence help is not merely information- "it's a relationship between the tutor and tutee and an essential component of the learning process" (Bruckman 1998). Giving and receiving help is part of a social connection. Beyond that, social interaction facilitates having role models, which are important. In MOOSE Crossing, one MUD member named Jim, actively chose to engage because he was "surrounded by peers who could program" which meant "he could give it a try knowing those friends could help him" (Bruckman 1998). Beyond that, he could

imagine being like them, which was an important component to him beginning and continuing his work.

3.4 Resnick's Collocated & Distributed Constructionism (Computer Clubhouse & Scratch)

Mitch Resnick also demonstrated the importance of social community in computing learning by developing computer clubhouses for young students. He said in order for them to become technologically fluent, they needed a type of immersion that facilitated living in "a digital community," which included interactions with technological equipment and people who knew how to explore, experiment, and express themselves with technology (Resnick and Rusk 1996).

Resnick also introduced distributed constructionism through creating the Scratch environment. Scratch is an online community that allows children to display and showcase their constructions, giving them the ability to discuss what they're working on, ways to better it, while allowing others to give feedback on it. Scratch enables distributed constructionism specifically through three categories of activities: 1) discussing constructions, 2) sharing constructions, and 3) collaborating on constructions (Resnick 1996). Scratch's uniqueness also comes from how it introduces beginners to coding through having them share their projects (and ideas for their projects) *online*. Both Resnick's Computer Clubhouse and Scratch facilitated collocated and distributed ways,

respectively, for learners to engage in supportive community based interactions, while creating their constructions.

To provide a bigger overview of the importance of support in environment, we also look at another scholar, Margolis, who cites the importance of supportive environments in subjects such as computing, and math and science, particularly in traditional collegiate environments. She discusses female students at Carnegie Mellon who attribute their undergrad survival to the support received from family and friends (Margolis and Fisher 2003). She also cites professors advocating for support in learning, referencing calculus professor Uri Treisman's sentiments on a supportive learning environment being critically important for the success of minority students in math and science (Margolis and Fisher 2003). Specifically, Margolis profiles Treisman's observations on distinctions between the high failure rate of African American students studying calculus at the University of California Berkeley and the high success rates of Asian American students (Margolis and Fisher 2003). Treisman notes that Asian American students form social communities where they help each other with math, compete at mastering the material, and generally support each other's learning (Margolis and Fisher 2003). In effect, Margolis' discussion of the benefits of supportive environments for math, science, and computing extend and ground the literature that discusses the benefits of having this kind of environment for students, and more specifically women (and minorities), when they engage in learning, particularly for technical subject material.

3.5 Recent Advances in Distributed Constructionism

Additionally, Parmaxi et. al builds on previous scholars' discussion on the importance of social support, which can effectively be termed social constructionism, and Resnick's distributed social support, also known as distributed constructionism (Parmaxi and Zaphiris 2014). They argue that the creation of the social web, with social technologies like Facebook, wikis, Dropbox, Google Docs, etc. all facilitate new dynamics and ways for the social constructionism to thrive (Parmaxi et al. 2013). Specifically, Parmaxi studies the construction of online artifacts in these media sites to show how learning through creating online artifacts collaboratively occurs in 9 stages:

- orientation
- brainstorming
- material exploration
- outlining, editing material
- revising
- peer reviewing
- instructor reviewing
- presenting
- publishing

Parmaxi's study shows how these types of online collaborative social platforms can "enhance learners' thinking and understanding of abstract ideas by relating them to a shared artifact."

It is important to reference these works in relation to our study for several reasons. At Hackbright, the influence and access of resources provided by the social setting, and the collocated and distributed network that emerged from it, was particularly notable. These resources facilitated continued engagement in the students' personalized projects and software engineering practices *even* after graduation. This is because the social network, while collocated *and* distributed, was able to avoid many of the temporal restrictions that would have impeded its survival in the past. Hence, the continued dialogue, interaction, and support among participants facilitated renewed discussions and opportunities for improvement professionally. These resources also provided existing support online and offline to those who needed it.

3.6 Co-Constructed Learning

The concept of knowledge or meaning emerging from the community is often identified as learning being co-constructed and has shown up in a several prominent educational theories, including Lave & Wagner's "Situated Learning" (Lave and Wenger 1991), Bruckman's "Community Supported Cooperative Learning" (Bruckman 1998), Slavin's "Cooperative Learning" (Marcu et al. 2010), Baxter and Magdola's "Learning Partnership

Model" (Kolko et al. 2012). These theories often appear in informal learning-environment models as scaffolding for how curriculum should be structured.

Moreover, the concept of learning being mutually constructed is also emergent in recent literature on informal and semi-formal learning engineering environments. In Kolko's *Hacademia*, a semi-formal program teaching non-technical students, engineering gave students opportunities to "validate their capacity to construct knowledge" by letting them define the scope and content of their work in the program (Kolko et al. 2012). Hackademia encouraged students to take an "active role in design" through "lack of specific external guidelines," pushing them to discover "what skills to learn and how to acquire them" (Kolko et al. 2012). This facilitated students "using their background as a learning framework to obtain additional knowledge," while relying on collaboration with other students to further skill acquisition (Kolko et al. 2012).

As described in more detail later, there was a similar experience at Hackbright Academy. Students had an informal, but semi-structured learning environment that presented fundamental programming challenges during the initial part of the course. This period of programming challenges was followed by instructor approved individualized students' projects where students picked an individual project and figured out what skills were needed and how to obtain those skills to complete the project. In the process, students at Hackbright gained experience building with vocational software technologies (Flask, Github, Parallax) as a corridor to showcase their ability to engage with newer technologies

in creative ways. Unlike students at universities or high schools, or even *Hackademia*, they also had access to industry professionals who, in most cases, understood and could help them contextualize the technologies they were learning to appeal to current industry standards. While some universities or schools have access to professionals who can help students with projects, what is noteworthy about Hackbright is the prolonged amount of access students had to industry mentors during their learning. Students engaged with mentors weekly, for half the program, sometimes extending their mentorship after the program was completed. This lead to continued professional development, during the project, and after.

Chapter 4- Related Works on Environments for Constructionist Learning

One reason constructionist environments are particularly relevant when discussing learning pedagogy for computing and engineering is their ability to simplify the abstraction prevalent in those disciplines. As one scholar put it:

> *"Students' learning progression is usually from the concrete to the abstract. Young people can learn most readily about things that are tangible and directly accessible to their senses—visual, auditory, tactile, and kinesthetic. With experience, they grow in their ability to understand abstract concepts, manipulate symbols, reason logically, and generalize. These skills develop slowly, however, and the dependence of most people on concrete examples of new ideas persists throughout life. Concrete experiences are most effective in learning when they occur in the context of some relevant conceptual structure."* (Dann and Cooper 2009)

In the following section, we review several noteable constructionist environments that have emerged since Papert introduced the concept of constructionism in digital technologies in 1980. It is important to note that these constructionist environments take many forms:

- some use simplified programming to create robots (Mindstorms),
- some use virtual networked interaction (Bruckman's MOOSE Crossing. Scratch) for students to learn and share projects,

- some simplify commonly used programming languages (Logo, Java, Python) to help learners adjust to learning programming concepts while creating projects, stories, animations, and robotics (LogoBlocks, Scratch, Alice, Google App Inventor).

There are also constructionist environments that are real life learning communities (Resnick's Computer Clubhouse, Shaw's MUSIC program) using people (mentors, teachers, etc.) to simplify learning challenges such as programming. Other environments are online constructionist platforms (i.e. Parmaxi et al.'s study) where people work together (i.e. Google Docs, Dropbox, Wikis, Github) to create and comment on shared artifacts.

In the following sections, we introduce how these environments relate to our study of Hackbright. We contrast similarities and differences in these environments. We note a shift in the way more recent constructionist environments have been designed to be less focused on "tinkering" (i.e. play for play's sake) and more centered on exhibiting sentiments posed by scholars such as Dewey and Papert to create more opportunities for real world use, while demonstrating flexibility for students to express broad interests and help others.

4.1 Logo & Lego Mindstorms

In the late 1960s, Papert invented the Logo programming language at the MIT Media Lab (Parmaxi and Zaphiris 2014). It allowed children to control and direct mechanical turtles to draw pictures through computer keyboard commands (forward, back, left, right, pen up,

pen down) (Parmaxi and Zaphiris 2014). In the process, children in this environment could control a turtle only by resolving problems related to angles, numbers, and graphical movements.

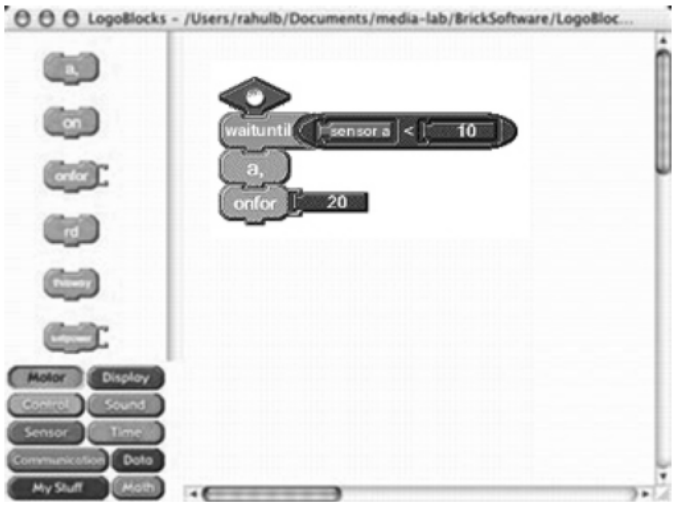


Figure 1: Snapshot of LogoBlocks Environment (reprinted from (Kelleher and Pausch 2005)**)**

In Figure 1, a screenshot of the LogoBlocks Environment shows the visual blocks children could piece together to create a functional line of code for a small program.

By using LogoBlocks as an "object to think with," children were active and self-directed learners in forming programming solutions. In the process, they demonstrated an adequate understanding of mathematical concepts in order for the drawings to materialize. Logo was inventive because it provided a play oriented methodology (such as creating a drawing) to incentivize children to learn abstract concepts they otherwise would not be familiar with.

In the process, the concept of children developing cognitive skills in mathematical concepts from play oriented activities became a more tangible technique for learning.

Despite Logo's innovation, it received some criticism. Research showed children using Logo had limited abstract knowledge after drawing activities (Parmaxi and Zaphiris 2014). Additionally, there was little potential for the knowledge children gained from using Logo to transfer into other kinds of learning (Parmaxi and Zaphiris 2014). The study also did not indicate that obtaining programming experience could translate into other domains (Parmaxi and Zaphiris 2014). Lastly, some believed the tool restricted other lines of thinking (Parmaxi and Zaphiris 2014). All these findings demonstrated that although benefits could be obtained from this technique in learning, there also were considerable limitations that needed to be addressed if this technique was to be more effective in the future.

Lego Mindstorms, a successor to Logo which also used its programming language, attempted to minimize some of Logo's earlier limitations. Mindstorms allowed children to play with software and hardware kits that contained Legos, sensors, gears, and motors. The kit could be used to make robot machines, showing that the Logo programming language could yield more advantageous and robust demonstrations of knowledge. A variation of these kits has since been released including: "Microworlds, StarLogo, and Programmable Bricks" (Parmaxi and Zaphiris 2014). Children could further customize these kits by connecting them to their computers to program controls for the robots. Later construction

kits, using Programmable Bricks (a large Lego Brick), could operate up to four motors at once and receive input from up to eight sensors. These kits were used for several reasons:

- "to help kids become more fluent and expressive with new technologies (and with "old" technologies);
- to help them explore important concepts (often in the domains of mathematics, science, and engineering) through their expressive activities; and
- to help them become better learners." (Resnick and Silverman 2005)

More recent constructionist kits have allowed children to use "traditional LEGO bricks for static, structural creations (such as houses and castles) or interactive constructions (such as animations in a virtual world or kinetic sculptures in the physical world)," but these kits have drawbacks (Resnick and Silverman 2005). Certain kits that have pre-arranged templates (like the Star Wars spaceship or Harry Potter castle) lead children to focus on "constructing the templates provided" so they can "learn by doing" without exploring the ideas underlying the construction of those designs (Resnick and Silverman 2005). This is a similar problem to some criticisms posited to the Logo programming language where learners could not transfer knowledge gained to activities outside the immediate activity they were engaging in.

In this sense, although Logo, Mindstorms, and other successor constructionist kits acted as a step in conceptualizing what could be done with these kinds of environments, it was hard to see what could be done with the knowledge gained from them outside of using them for hobbyist practices. As indicated by studies on children using Logo, it wasn't clear what children could do with the knowledge they gained once the prescribed activity was completed. Similarly, Lego Mindstorms and succeeding construction kits, allowed for students to tinker with robotics, sensors, lights, etc., but not necessarily to build upon the learning from that tinkering (as is prescribed in "bottom-up learning"). This was primarily because it was not completely clear how what was learned from these environments could be used in a real life context (Utting et al. 2010).

Resnick's Computer Clubhouse, Hackbright Academy, and more recent college courses featuring a CS 0 course (Klawe Harvey Mudd curriculum, Karakus's Google App Inventor) have all aimed at addressing this issue by developing new constructionist environments that are grounded in real life application. With the growth of ubiquitous devices such mobile phones, these newer constructionist environments simplify programming (through platforms such as Github, Scratch, Google App Inventor), but allow for creating and designing technology with real world applications in mobile or computing devices. They enable individuals without prior programming experience to envision tools they always wanted but did not think they could create, while encouraging individuality and empowerment through designing and creating technological tools. More importantly, the

tools are ubiquitous enough for learners to begin to understand the breath of their use. Hence, these *real life* tools allow for learners to understand the context of where and how the knowledge they learn can be applied.

In addition, Hackbright's constructionist environment, with a particular focus on women who are more socially oriented learners, uses not only active and self-directed learning, but also collaborative paired exercises (and collocated spaces), to facilitate learning programming concepts. This environment, in comparison to constructionist environments like Logo or Lego's Mindstorms kits, resolves the need many women have to engage in constructionist activities that are more collective and social. Moreover, it allows them to help each other, while showing the potential for helping others, in the real world.

4.2 Resnick's Computer Clubhouse

While looking at constructionist environments, it's also important to discuss Resnick's Computer Clubhouse. Resnick's Computer Clubhouse showed a new model of a learning community that changed the traditional practices of learning (particularly in a computer lab). It allowed "inner city youth" to become designers and creators of computer based products as opposed to just consumers of them (Resnick and Rusk 1996). At the Computer Clubhouse, youth created many different types of projects including video games, digital stories, interface designs, and digital art projects.

There are four core principles of the Clubhouse educational approach:

- supporting learning through design experiences (students creating their own computer games instead of playing them),
- helping students build on their own interests (learning to use Photoshop to augment a student's comic book drawings),
- cultivating an "emergent community" (this builds on the concept that for young people to be fluent in a language, they must be immersed in it and have a space for that immersion)
- creating an environment of respect and trust (Resnick and Rusk 1996).

Hence, Resnick's computer clubhouse inherently reinforced and updated Papert's vision of a *technological* samba school (Parmaxi and Zaphiris 2014). "Technological samba schools," which Papert discussed in his 1980 book Mindstorms (Papert 1980), reference Papert's desire to merge technical learning with the culture observed in samba schools in Brazil. Papert wanted technical learning to include (Papert 1980):

- students being self-motivated
- richly connected to popular culture
- focused on personally meaningful projects
- in an environment that is community based

In samba schools, a community of people of all ages gather together to prepare a presentation for carnival. Papert describes his observations by saying:

> *"Members of the school range in age from children to grandparents and in ability from novice to professional. But they dance together and as they dance everyone is learning and teaching as well as dancing. Even the stars are there to learn their difficult parts"* (Papert 1980)

Resnick's Clubhouse model realized much of Papert's vision in several ways. Computer clubhouses were community based, focus on expressive projects that were individualistic, and self-motivated because students designed their own projects. Arguably they were also connected to popular culture because students would take hobbies they had (i.e. photography or drawing) and blend that interest with current software that could refine or augment their creations in new ways (i.e. through Photoshop) (Resnick and Rusk 1996).

Hackbright, like the Computer Clubhouse, also makes a point of creating an environment that encompasses a lot of what Papert envisioned in a technological samba school. It uses:

- current and popular technological culture,
- encourages self-motivated learning through personally meaningful projects,
- is community based for current students and alums (during lab exercises, events, and even after the program)

Additionally, the goals that Resnick aimed to achieve at the Computer Clubhouse are also goals that are self-evident at Hackbright. At Hackbright, the goals include:

- a culture of respect and trust for learning and an emergent community that can be readily seen in Hackbright's "females in tech" events,
- open demo events for Hackbright students to showcase their projects to the community,
- cross-class reunions/parties, and other types of social/career development events (to develop and maintain unity amongst past and present students),
- an undertone of support, complete openness, and vulnerability during the course (e.g. no questions are dumb questions).

Also like Resnick's Computer Clubhouse, Hackbright students design and create a variety of software projects ranging from 3-D rendering projects using Parallax, to language translation web apps, to mobile Twitter-like apps that provide updates for special groups like school teachers. There are differences in these environments however. Hackbright's environment is distinct because it:

- has an all-female student body,
- utilizes industry mentorship that includes distributed constructionist activities (help through chat, emails, Skype),
- includes career training with a focus on salary negotiations/interview preparations,

- engages in online artifact creation in software project sites such as Github.

4.3 MOOSE Crossing

While discussing community based constructionist environments, it's also important to look at Bruckman's MOOSE Crossing. MOOSE Crossing (1997) is a text- based MUD (multi-user dungeon) "networked-programming environment for children" (Kelleher and Pausch 2005). It uses "an object-oriented scripting language" (Kelleher and Pausch 2005) to create spaces and characters in a make-believe textual world. Learners create rooms, laboratories, castles, helicopters, and other spaces (or characters) similar to those found in text adventure games, sometimes with secret passages that other learners can explore. Once projects are completed, any learner in the MOOSE Crossing environment can interact with the project.

```
> @kids #445

generic portable room(#445) has 124 kids.
Yellow Cab(#296)
Generic Neo-Scrabble Board(#1439)
MediaMOO TV Helicopter(#1659)
The Bob Marley Community Media Bus(#3655)
Generic Train(#4559)
Sound Proof Room(#3136)
an emacs window(#4607)
'77 Jeep Cherokee(#5731)
goldfish bowl(#4761)
portable hole(#6165)
The Autonomous Drone(#7815)
StarkNet School Bus(#7930)
Box of Delights(#8233)
Bible Study Room(#4014)
Generic Cards Room(#8757)
Jack's Laboratory(#3401)
dictionary(#6303)
water(#9225)
tiny envelope(#9566)
An Igloo(#8999)
an aluminum mailbox(#5449)
[etc.]
```

Figure 2: Snapshot of MOOSE Crossing Environment (reprinted from (Bruckman 1998)**)**

Figure 2 illustrates this activity. Reading from the top of Figure 2, once a learner in the MOOSE Crossing environment enters a room (room #445), they see the number of "kids" in the room (124) and all the objects (i.e. Yellow Cab), characters, and sub-rooms (i.e. Generic Cards Room) that are connected to it. Learners can then press a command to enter an object or room and interact with other learners in the room or view the scripts used to create that object.

Additionally, this environment lets beginners look at how these spaces, characters, or passages are created. The scripts controlling any object can be seen by users entering the environment. Learners can also chat with others logged onto MOOSE Crossing. Many times learners have their own projects and are open to chat and/or display their projects. Although most learners work alone on projects, a project turns into a vehicle for others to use as an example and begin programming. Learners can ask others for help or advice and in turn get a place that provides role models and positive feedback for users of the system.

Hackbright has several similarities and differences to MOOSE Crossing. In both environments, learners rely on each other (peer learning model is obtained in Hackbright through pair programming) and showcase their artifacts (at Hackbright it's done through Hackbright demo events). Both also have a networked communal interaction, but with Hackbright it is done through:

- an email list serve where people share social/professional information (including open jobs at their company or invitations to study an emerging technical practice)
- actual networking events and constructionist activities supported through industry mentors working using collocation and computer supported cooperative learning practices (i.e. over the shoulder learning)
- networked interaction including real life software project sites (such as Github) which showcase technical skill for vocational and professionalization purposes.

4.4 Alice

Alice is another constructionist environment worthy of discussion, not for its community based approach, but because of its ability to harness technological practices for learning coding, while expressing individualistic interests and creativity. Alice allows children and adults to create characters, objects, and engage in storytelling while learning to program.

Created by Randy Paush at Carnegie Mellon, Alice uses "a programmable 3D-authoring tool to make interactive 3D-graphical worlds" accessible to middle school, high school and college-level, non-science majors (Kelleher and Pausch 2005, Dann and Cooper 2009). Similar to MOOSE Crossing's use of a text-based virtual-reality world to motivate learning programming, Alice uses animations "to teach students problem solving and algorithm building" using simplified versions of mainstream programming languages (Pyton, Java) modified based on user recommendations (Kelleher and Pausch 2005).

Alice 3 code editor, scene editor, and runtime displays.

Figure 3: Snapshot of Alice Environment (reprinted from (Kelleher and Pausch 2005)**)**

Figure 3 above shows a screenshot of the authoring tool being used to create part of a scene with characters.

Alice has had several updates from Alice98, to Generic Alice, to Alice 1,2,3 (Storytelling Alice). The most notable changes from older Alice versions and the most recent Storytelling Alice are:

- "Storytelling Alice provides high- level animations inspired by girls' storytelling goals (while Generic Alice provides animations inspired by common 3D graphics transformations),

- In Storytelling Alice, users' programs animate simple stories (while in generic Alice, users' programs cause 3D objects to move, turn, and resize).
- In Storytelling Alice, the gallery includes characters with custom animations….while 3D objects in the Generic Alice gallery do not include custom animations." (Kelleher and Pausch 2005)

Alice has several similarities and distinctions from Hackbright as a constructionist environment. One similarity is both environments promote more gender diversity by finding "female friendly" versions of artifact expression. Hackbright is distinctive however because it is a physically collocated rather than virtual. Additionally, while Alice is a 3D authoring tool using modified versions of Java and Python (like Logo, Scratch, and construction kits like Mindstorms) to promote learning, Hackbright uses modern distributed constructionist technologies like Github and social media such as Skype or chatting apps like Google chat to obtain help from mentors, friends, etc. during personal artifact creation. Moreover, Hackbright is a community based constructionist environment (like Scratch, MOOSE Crossing, Resnick's Computer Clubhouse), while Alice is more of a tool that is used for students to learning to program while coming up with creative ways to express their learning through animations. Lastly, Hackbright's constructions have goals that instill change and so they can join the technical dialogue. Students are actively trying to create technologies that can help others. For example, some Hackbright student projects include:

- building a mobile app for handicapped individuals to use for taking Philadelphia transportation
- designing a mobile app that uses the accelerometer sensor in phones to measure specific frequencies occurring during Parkinson's trembles.

4.5 Scratch

Scratch (http://scratch.mit.edu) is a virtual learning constructionist environment (similar to MOOSE Crossing) where learners can program "interactive stories, games, animations, and simulations in a 2-dimensional environment" (Resnick et al. 2009). Creating a program requires "snapping graphical programming blocks together into a script, like snapping LEGO programming bricks together "(Kelleher and Pausch 2005; Resnick and Rosenbaum 2013, Resnick et al. 2009).

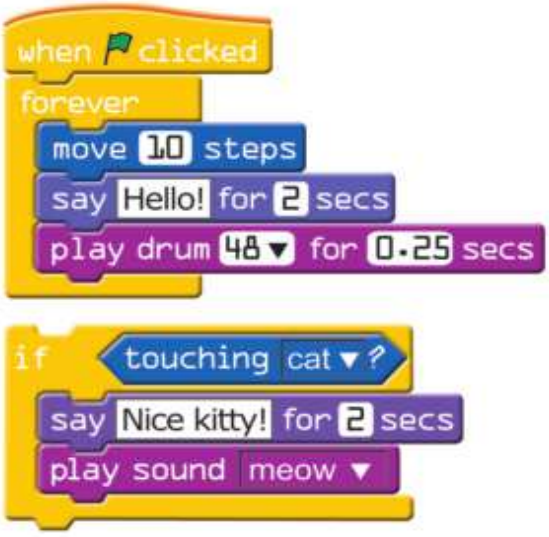


Figure 4: Snapshot of Scratch Environment (reprinted from (Resnick et al. 2009)**)**

Figure 4 shows the use of these graphical blocks (on the left) being put together into several lines of code that create a simple “Nice Kitty” program.

The difference between virtual networked environments such as Scratch (and Google App Inventor) and more physical constructionist spaces (Resnick’s clubhouse, Hackbright Academy) is that in the latter learning to program is not represented in snapping together LEGO programming bricks, or graphical blocks online, but in using other people as tools to help students learn programming fundamentals. People that become the tools for learning may be classmates, instructors, TAs, tech series speakers, industry mentors, or even friends/family/significant others. Additionally, in Hackbright’s lab exercises specifically, the programming building blocks are not interactive stories or animations, but exercises that teach fundamentals of programming needed for students to go on and create their own individualistic real world project.

4.6 Google App Inventor

Google App Inventor allows beginners to create mobile apps or personalized software for their phone. It is different from Alice and Scratch because instead of motivating students to program through storytelling and multi-media animations in a visual environment, it allows them to create apps that augment their actual reality. It uses a similar blocks language to Scratch, which have proven successful with both children and college students (in a university course) (Karakus et al. 2012).

Figure 5: Snapshot of Google App Inventor Environment (reprinted from (Kelleher and Pausch 2005)**)**

As shown in Figure 5, a simple program can consist of several blocks being placed together horizontally (to form a line of code) and vertically until sufficient to create a functional program.

Additionally, Google App Inventor allows students, while learning to program a mobile application:

- to use and process SMS texts (as part of the application they are building),
- to work with the GPS location sensor of the phone,
- to scan barcodes, and
- to communicate with web APIs.

It also enables students to use technological advances in the mobile industry to further projects stemming from their learning. That enables students to demonstrate greater chances at innovation during project design that are concurrently applicable to real world needs.

Hackbright's constructionist environment demonstrates many of the benefits displayed through the use Google App Inventor in schools and college courses. Hackbright encourages beginners to use platforms like mobile devices or Github to illustrate personalized and meaningful projects. Additionally Hackbright, like the distributed constructionism outlined in Parmaxi (Parmaxi and Zaphiris 2014), allows students to work on personal artifacts through software project sites like Github to share ideas, while constructing external and shared artifacts. Hackbright is different than Google App Inventor because it's not a software constructionist environment, but rather a learning community that uses many methods and tools to simplify learning including;

- teacher sessions/lectures,
- collaborative pair programming lab sessions,
- distributed and collocated community-based interaction through an email list serve,
- interactions with industry mentors online and in person, all while engaging in a supportive and safe environment.

4.7 Challenges with Constructionist Environments

Some challenges with the constructionist paradigm involve the concept of tinkering as a stepping-stone to learning (Utting et al. 2010). The concern is to create an environment where the tinkering does in fact facilitate a jump to the next level of comprehension for a particular mathematical, scientific, or engineering concept. In other words, the tinkering must lead *somewhere* (must be bottom up learning where learning builds on knowledge gained from tinkering). As Dewey and Papert suggest in their works, a teacher's role is to design curriculum that allows students to build upon whatever mistakes they discover while tinkering and therefore develop an understanding of a particular concept in question.

At Hackbright, some challenges from the constructionist environment were visible. Constructionist building-upon-mistakes was effective in class sessions with:

- smaller classroom sizes so the teacher could keep track and direct students more effectively and
- teachers who were familiar with computing concepts the student was using in the project.

The omission of either of these characteristics in a constructionist environment led to breakdowns in communication and learning, while effecting the efficacy of the resources available to students.

Moreover it is important to note that recent constructionist environments like Alice, Scratch, Google App Inventor, embrace "girly" topics by emphasizing individuality and empowering users. This empowerment is also visible in Computer Clubhouses, MOOSE Crossing, and Hackbright. These constructionist environments are more socially oriented, more interested in context, and making a difference in society. They divert away from earlier constructionist environments (Logo, Mindstorms, construction kits, etc.) that were more focused on play for play's sake, and hence deemed to be more masculine (Utting et al. 2010). Constructionist environments like Google App Inventor, and Hackbright particularly, are more "realistic," using modern ubiquitous technologies like Github or mobile phone apps, while creating opportunities for change and helping others. Moreover, Hackbright as a constructionist environment is distinct because its ultimate focus is not just education, but using these technologies to promote rapid professionalization.

What becomes most relevant from this discussion is the progression of constructionist environments from more tinkering-based (Logo) environments to ones more focused on constructing objects that may lead to immediate employment (Hackbright).

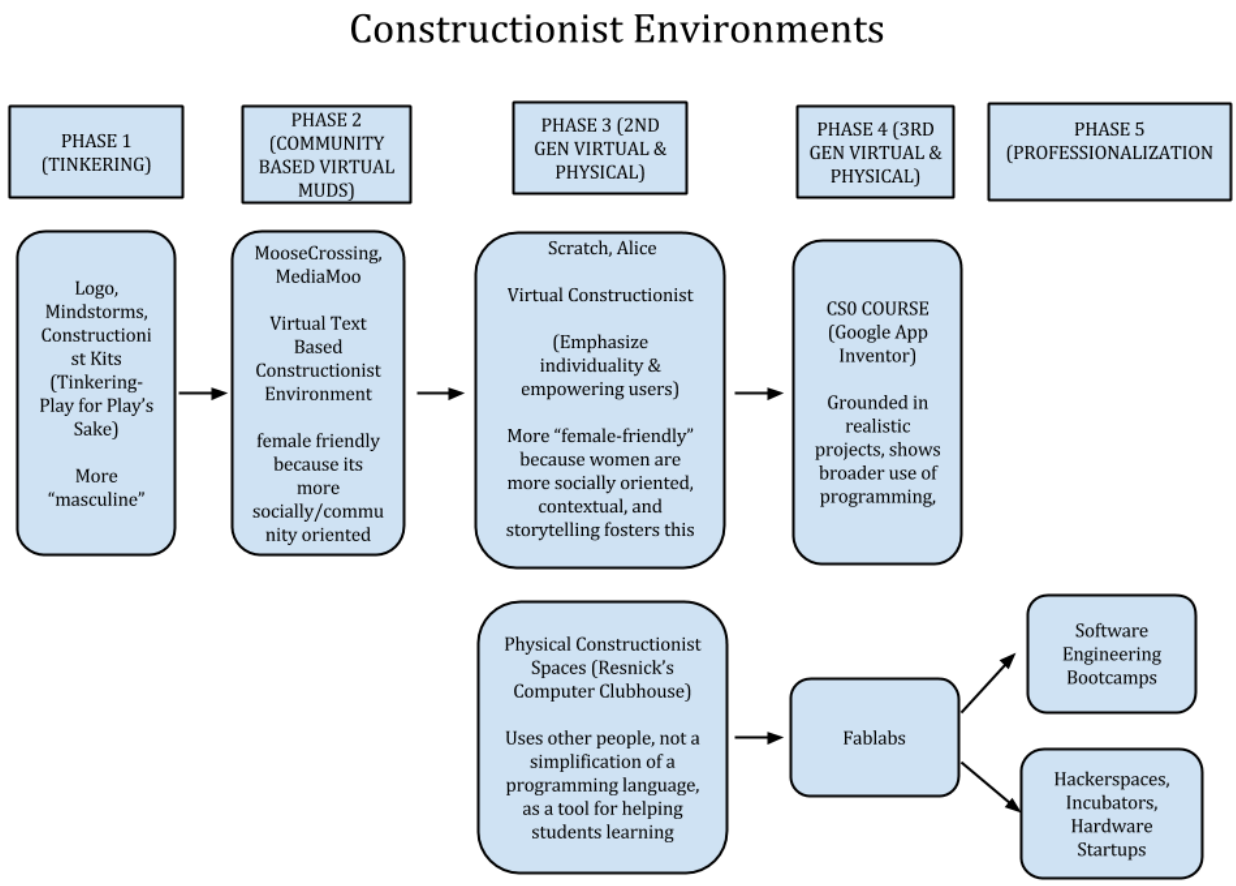


Figure 6: Phases of Constructionist Environments (From the 1960s-Present Day)

In Figure 6 above, we aim to visualize the progress of constructionist environments over the last several decades. Initially, constructionist environments were designed to facilitate tinkering (Phase 1). Environments like Logo, Mindstorms, and other construction kits were designed to have children or learners interested in computers or technology play in order to learn mathematical or computing concepts. The difficulty with some of these environments was children might try to just create a drawing (Logo) or build a robot (Mindstorms) without necessarily understanding why they were doing it or how it was

transferrable. Also these environments did not necessarily foster social interaction or understanding the application beyond the activity prescribed in the environment.

In Phase 2 in Figure 6, a shift in constructionist environments is made where networked interaction can occur among students trying to learn in a virtual world. Despite these environments only being text-based, eliminating visual demonstrations of constructed learning, these environments nonetheless allowed for children (and interested adults) to share and learn from each other while displaying artifacts. This phase illustrates a shift to a constructionist activity that is more social, community-based, and paves the way for constructionist environments that include emphasize those aspects while incorporating more visually engaging projects.

In Phase 3, constructionist environments shift again. They include *both* virtual and physical constructionist environments. The virtual environments that are now available (Scratch, Alice) allow for students to use storytelling (Alice) or interactive media (Scratch) such as video games, birthday cards, and interactive tutorials. Physical constructionist environments like the Computer Clubhouse allow students to come together in a shared spaced and work together, learn from each other, and augment hobbies (photography or drawing) with technical skill (creating drawings in Photoshop). All environments in Phase 3 allow for social interaction, individual and creative expression, but they also open the door for students to begin creating projects that are more oriented in real world application. Since people use Photoshop and play video games the real world, there is a

progression for constructionist activities to become more contextually relevant to current technical practices.

In Phase 4, virtual constructionist environments become even more contextually relevant to the world today. A shift is made so that building blocks software does not just build games or birthday cards (Scratch) or stories (Alice), but can build mobile applications (Google App Inventor) that can be used for a variety of broad applications. This change facilitates more inclusion of women since they are typically drawn to environments where they can see how an application would be useful to them or others. Furthermore, several courses in schools *and* colleges use Google App Inventor to teach non-majors programming. In physical constructionist environment, "Fablabs" are designed to foster "making" in learning environments like schools. This shows the progression of physical constructionist environments moving from a hobbyist activity (in after school) to an activity in schools.

In Phase 6, we see the final shift in constructionist activities from hobbyist to school and then professional areas. Constructionist environments not only create the *products* that emerge from hardware startups or makerspaces, but they create *people* who become professionals (i.e. software engineers in bootcamps). That is, bootcamps professionalize students: they become the products (along with their projects which are used to affirm their professionalization and readiness for employment).

Chapter 5- Related Work on Women in CS, CSCL/CSCW, Pair programming, Collocation

Now we turn towards other streams of literature that are relevant to our analysis of Hackbright. We discuss related works from computer science research, computer supported cooperative learning practices, computer supported cooperative work practices, pair programming, and collocation practices.

5.1 Inclusion of Women in Computer Science

"Many assume that programming a computer is a difficult activity that should be undertaken only by the technically educated elite; it's not the province of a mere building contractor or humanities majors, or women; technology increasingly surrounds our everyday lives, but most people can't imagine themselves having meaningful control over it. For girls and women, the problem is compounded: they may fear success with a computer as much as they fear failure."

-Sherry Turkle (Bruckman 1998)

Several academics have addressed the concern to bring more women into technology, science, and engineering disciplines at the college level through creating and implementing new curriculum. One change was the development of a CS 0 course to facilitate interest in software development to students without prior computing exposure (Karakus et al. 2012; Klawe 2013; Margolis and Fisher 2003). Some institutions also changed the marketing and structure of courses to include more real-world applications (as opposed to theoretical

problem sets), while changing course titles and names to reflect a more modern approach to teaching (Margolis and Fisher 2003). Variations of approaches that are in this vein are now in place at prominent colleges such as MIT, Georgia Tech, and Harvey Mudd (Forte and Guzdial 2005; Margolis and Fisher 2003; Klawe 2013).

Despite there being discussion of these advances, which have led to expansion of female enrollment in computer science departments in colleges (Milam 2012), and discussion of initiatives supporting engagement of girls in middle schools and high schools in engineering activities (Boyer et al. 2014; Kuznetsov et al. 2011), the existing body of research provides little discussion, if any, on post-collegiate women transitioning into the computer science field through software development bootcamps, or other informal learning environments.

It is the aim of this research to present a qualitative study that looks at one all-female software development bootcamp, Hackbright Academy, whose main goal is to bring more women into computer science jobs. Hackbright does not call itself a bootcamp, however. It describes its course as a "software engineering fellowship" to convey the importance of community and sharing, if not to demonstrate the high caliber of its program. In this study on Hackbright Academy, we evaluate its pedagogical practices, students, and graduate success rates, while providing educational design considerations for promoting diversity in computing and engineering disciplines.

5.2 Computer Supported Cooperative Learning/Computer Supported Cooperative Work Patterns

Our study of Hackbright extends research in aspects of Computer Supported Cooperative Learning, including over-the-shoulder learning (M. B. Twidale, Wang, and Hinn 2005; M. Twidale 2013). Over-the-Shoulder Learning (OTSL) is a type of learning that enables shared context (Miller et al. 2014). Shared context is where "the helper understands the task the learner is trying to do and the learner's goals for doing it" (Miller et al. 2014). OTSL sometimes requires organizational changes so that giving help to colleagues leads to effective demonstrations of benefits of the help. Our study of Hackbright hopes to demonstrate the positive value of giving help, particularly in a new setting, such as a software engineering bootcamp.

Our study of Hackbright also hopes to expand on Berlin and Jeffries work in apprentice learning (Berlin and Jeffries 1992). In this study, graduate students and their mentors were observed in their computer science labs. One finding was that the occurrence of incidental learning, where events requiring conflict resolution, enabled learning between apprentices and their mentors (Miller et al. 2014). Apprentices also had to limit their use of mentor's time and therefore developed strategies to do so while maximizing learning. By Hackbright having many of the students help each other, they minimized their reliance on apprentice learning provided by teaching staff or actual industry mentors. Nonetheless, some

apprentice learning occurred during whiteboarding sessions between mentors and mentees (or between teachers and mentees). Additional instances of apprentice learning occurred through mentees seeking help on their artifacts or through getting professional development advice from mentors or teaching staff.

5.3 Pair Programming

Williams et al. describes pair programming (Cockburn and Williams 2000) as:

> *"In pair programming, two programmers jointly produce one artifact (design, algorithm, code). The two programmers are like a unified, intelligent organism working with one mind, responsible for every aspect of this artifact. One partner, the driver, controls the pencil, mouse, or keyboard and writes the code. The other partner continuously and actively observes the driver's work, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications. The partners deliberately switch roles periodically. Both are equal, active participants in the process at all times and wholly share the ownership of the work product, whether it is a morning's effort or an entire project* (Cockburn and Williams 2000)*."*

In Williams et al., one study found there were several categories where pair programming proved beneficial (Cockburn and Williams 2000):

- *"better economics (reduced cost of defects),*

- *improved satisfaction (programmers experience more enjoyable than working alone),*
- *faster problem solving (pair relaying)*
- *improved design quality b/c of continuous reviews (more efficient programs from shoulder to shoulder learning technique),*
- *improved learning, about the system and about software development (line- of-sight learning),*
- *better team building and communication (the people learn to work together and talk more often together, giving better information flow and team dynamics)*
- *increased staff and project management (reduced risk of staff loss because more staff familiar with the code)."*

The study also went on to discuss pair programming as allowing for learning through "expert in earshot," "legitimate peripheral participation," and "line-of-sight" learning.

Additional findings suggested that pair programming allowed for (Cockburn and Williams 2000):

- *"learning to work together,*
- *sharing problems and solutions efficiently (better teamwork)*
- *developing ways to communicate more easily and more often (raised communication bandwidth and overall information flow within the team).*

Other more recent studies have gone on to discuss additional benefits and drawbacks of pair programming. Wills et al. looks at the efficacy of putting together students on self-ranking questionnaire responses about confidence with material (Wills, Davis, and Cooke 2004). Other studies looked at the effects of certain factors in pair programming efficacy such as: learning styles, programming self-esteem levels, work ethic and time-management skills, differing personality types, similar perceived skill levels and similar actual skill levels. In addition, the research was inconclusive on how to most effectively match a pair (Cliburn 2003).

Carver et al. suggests that the ability to learn from one's partner, the lowering of student frustration, and the improving of communication skills are definite advantages that come from pair programming (Carver et al. 2007). Williams and Kessler also suggest pair programming is beneficial because it encourages shared ownership, since both partners participate and contribute (Williams and Kessler 2000). It also keeps partners more concentrated on the work to be done and helps partners expand each other's aptitude in programming (Williams and Kessler 2000). However, studies have also noted challenges to achieving success with pair programming, including mismatched schedules (Bevan, Werner, and McDowell 2002; Cliburn 2003) and pair incompatibility (Bevan, Werner, and McDowell 2002; McDowell, Hanks, and Werner 2003; McDowell et al. 2002).

According to NCWIT's 2007 report on pair programming, there are specific benefits for pair programming on women. These include that it (NCWIT 2015):

- *"increases likelihood of students (particularly women) declaring a computer science major;*
- *grows the number of students in the computer science major one year later, (in contrast to non-paired classmates)*
- *lowers the "confidence gap" between female and male students and raises programming confidence of all students;*
- *initiates better-quality student programs compared to non-paired peer programs."*

Our study extends the findings from these previous studies by presenting more qualitative information regarding potential advantages and disadvantages of pair programming, with specific emphasis on the pair programming practices with the adult female population, in an informal constructionist environment.

5.4 Collocation

Teasley et. al. studied companies putting teams in "war rooms for productivity enhancement," through a field study with 6 teams. They examined activity, attitudes, use of technology and productivity (Teasley et al. 2000). Teams in war rooms "showed doubling of productivity" (Teasley et al. 2000). Also they noted that teams had "easy access to each other for coordination of their work, for learning, and work artifacts remained visible to all" so that people could be aware of everyone's process on their tasks (Teasley et al. 2000).

At Hackbright, although the environment is a constructionist learning environment, many parts of it replicate or prepare its students for workplace practices in technology companies like the one studied in Teasley's research. Hackbright students often work in paired teams and have to explain direction, progress, and their approach to lab exercises during the first five weeks of the course. Many companies today still use pair programming exercises in the workplace to increase productivity and reduce error rates in code. Additionally, Hackbright students engage in scrum meetings (during the last five weeks of the course), which are a common practice in many technology companies today. The difference is that Hackbright students update their teachers, TAs, and other students on individual progress, goals, and conflicts they might need to address/resolve, instead of co-workers and managers.

Therefore, when we analyze the collocation practices at Hackbright, with Teasley's work in mind, we see the benefits of radical collocation on the productivity of students. The instructional co-founder of the Hackbright program, Christian, noted that participants mentioned that just having mentors collocated with them enabled them to feel more supported and encouraged to step up and do the work, even if they did not actually end up asking their mentor many questions.

In Covi's et.al's study of the collaborative habits of teams in 9 U.S. companies who had dedicated project rooms, research showed that team members using "dedicated project rooms reported clear advantages" (Covi et al. 1998). These advantages included "increased

learning, motivations, and coordination" (Covi et al. 1998). Findings also suggested buildings needed "to support features of collocated teamwork such as shared display and awareness of team members activities" (Covi et al. 1998).

Chapter 6- Research Methods

In this study, we used multiple methods including participant observation, shadowing, and semi-structured interviews to understand the learning practices of 15 adult female students in one 10-week software engineering fellowship in California.

We conducted observations of project demos of 5 female students upon completion of the software engineering fellowship. We shadowed 3 participants at the software engineering fellowship field site and at a project showcase at GitHub headquarters. We also conducted semi-structured interviews with 15 software engineering fellowship graduates and 2 interviews with both of the original co-founders of the Hackbright program. Interview participants included graduates who had just completed the fellowship, along with graduates from earlier class sessions, most of whom were already working in the software engineering industry.

Our interviews focused on motivations for joining Hackbright, including previous background before joining, interactions with peers, instructors, and mentors provided during the fellowship, educational environment, career development, and individual projects. All interviews were audio recorded and transcribed.

6.1 Hackbright in Practice

Each software development cohort session lasts 10 weeks and is taught primarily in Python. Daily sessions occur Monday through Friday from 10 am to 6 pm. During the first 5 weeks, students attend class sessions with instructors discussing computing fundamentals by a main teacher, Christian (an instructional co-founder who has since left the program), and four other supporting instructors. The instructional design has since changed in the last 2 cohorts so that there are 3 instructors, 3 teaching assistants, and 3 instructional developers.

The schedule for the first five weeks features lectures and pair programming. Each day's schedule from 10 am- 6 pm is as follows:

- 10 am- Morning Lecture
- 11 am- Pair Programming exercises begin
- 1 pm- Lunch Break
- 2 pm- Lightning Tech Talk (given by students on topic of their choosing)
- 2:15 pm- Afternoon Lecture
- 3 pm- More Pair programming exercises
- 6 pm- Session Ends

Typically lectures total about 2 hours each day (one from 10 am- 11am, one from 2:15 pm- 3 pm) before the cohort members choose their pair programming partners. Hackbright strongly suggests picking a different pair programming partner during each lab exercise (this practice is known as "promiscuous pair programming"). Also during the first five weeks, students are to give a brief presentation on a technical subject of their choosing, called a "tech talk," lasting about ten minutes, in order to demonstrate more familiarity with current technical trends in the industry.

Each day they engage in several programming exercises (some of which can be found in Github under the instructor's Hackbright curriculum repository, https://github.com/hackbrightacademy) in order to develop knowledge of the computing fundamentals needed to construct their personal projects in the second five weeks of the course.

The schedule for the remaining five weeks of the course, Weeks 6-10 is as follows:

- 10 am- Scrum meetings (Agile development technique)
- 11 am- Programming for personal projects
- 6 pm- Session Ends

After the initial 5 weeks, students choose their own individual projects with the approval of instructors. Daily sessions began with scrum meetings, which are a common in agile development practices in the software development industry. Scrum meetings last for one

hour and allow students to report on progress and troubleshoot student project roadblocks. These meetings allow for each student to become familiar with the work of their peers, while enabling the staff and other students to help resolve concerns and learning challenges that may arise.

Students can seek the assistance and guidance of instructors, teaching assistants, instructional developers, their peers, and industry mentors to help them develop their projects. Many of the participants interviewed sought help from friends and or significant others in figuring out the design and implementation of their project.

We refer to Hackbright as an informal learning environment since there is no academic credit or academic degree conferred upon participants. Although there are instructors who utilize lecture-based curriculum for the initial weeks of the program, the lectures are free form:

- there are no formalized lesson plans
- lectures vary depending on questions raised.

Hackbright's program also features social events such as showcases, fieldtrips to companies like Google, Pinterest, Intuit, Microsoft, etc., and other events at local tech companies. Many companies welcome the women and discuss their need to have women represented in the company.

Lastly, upon finishing the course, Hackbright students showcase their projects at a career day with 20-30 industry partners. During career day, they demo their projects and conduct rapid-fire interviews with representatives from partner companies. The participants' experience in the program and exposure at these events, rather than a degree, becomes the gateway to obtaining a position as a software engineer in industry.

Chapter 7- Results

7.1 Demographics of Accepted Hackbright Interviewees

We began our study evaluating the demographics of the 15 female interviewees. We looked at motivations for joining, their age, educational/professional backgrounds, and additional commonalities (including outside interests).

7.1.1 *Motivations for Joining*

Participants reported that they joined the software engineering fellowship for several reasons:

- wanting to create a bigger impact in their lives and work (p1, p3, p5, p7, p8)
- wanting to build things (p1-p4, p7, p13)
- wanting to find a job that is fulfilling and enjoyable (p1, p2, p9, p15)
- disempowerment/ lack of mobility in career choices upon collegiate graduation (p1-p8, p11, p13, p15)
- wanting to be financially independent (p1, p3, p6, p7, p9)
- wanting to learn complex technical material in a place where they felt "safe."

7.1.2 Age/Education Demographics Before/After Hackbright

Interviewees ranged in age from 20 to 42. The average age of participants interviewed was approximately 28 years old. Many interviewees had a graduate degrees (6), some had a bachelors degree (7), while others had completed partial college coursework (2).

7.1.3 Most Participants Knew a Hackbright Graduate Before Applying

Most participants had a friend in Hackbright or had spoken to a Hackbright alum before making the decision to apply for the program (p1, p2, p3, p6, p7, p10, p11, p13, p15). They reported that meeting Hackbright alums who had similar interests, thought patterns, and had completed the program, while finding the industry work to be fulfilling, was important in their decision-making to apply to the program. Specifically, one participant noted that she had "more in common with these people [alums] than anyone else" (p11). Also she indicated that referrals might "carry some weight" since she had a friend who was an alum from the third Hackbright class (p11). Another had a close friend who had gone through the program that was a "big source of support" because she "could turn" to her "anytime" (p6). Other participants knew each other before applying, talked to each other about applying, and ended up joining the same cohort (p10, p15). Another participant turned to a previous fellow to get a sense of the program, asking "how she felt going through the process." She reported that this conversation helped her because she "felt like she wasn't alone" in being "scared, excited" during the program (p7).

7.1.4 Previous Exposure to Technology through Schooling, Family, Friends, Employment

Six participants had previous exposure to technology through their work environments or previous schooling which provided enough familiarity to consider transitioning into software engineering career (p1,p5, p7, p10, p12, p13). Five participants had friends or family in the software industry (p2, p4, p6, p9, 14). Three had a science or math background that facilitated an easier transition into software development (p4, p5, p14).

7.1.5 Employment Before and After Hackbright

Although at the time of interviewing, 10 out of the 15 of the participants were not currently employed (1 was in school, 1 was freelancing, and 8 were still interviewing). Follow-up surveys conducted approximately one year later showed that all participants were working in the software development industry, primarily as software engineers or data scientists. One now works as an instructor at Hackbright Academy, 2 are working as data scientists at Keen.IO and Change.org, while the remaining 7 are working as software engineers at companies such as Heroku, New Relic, Stripe, SurveyMonkey, and Crittercism.

7.1.6 The Best Learners Are Teachers?

Preliminary demographics revealed that many of the interview participants (8 out 15) were previously teachers, camp counselors, or coaches (p1, p2, p3, p5, p6, p8, p11. p12). During semi-structured interviews, both co-founders (p16, p17) indicated the program looked for candidates with:

1) a passionate interest in learning to code

2) an ability to learn and teach a skill (both termed this skill as demonstrating the potential for how good a programmer could be).

At the time of the study, Hackbright used a unique selection process for interviewing and selecting participants. During the previous cohort application process prior to the study, Hackbright had to sift through 400 applicants for only 30 available slots (8% acceptance rate). Candidates that were selected for an interview had an initial interview by phone or Skype. Each applicant was screened for her ability to break down skills in a subject that she had become an "expert" (p17). Phone interviewers were looking to see how comprehensively and clearly interviewees could explain a subject, and if done effectively, she would move on to the next round of interviews (p16, p17).

Also 5 out 15 of interview participants had husbands, partners, boyfriends, or fiancés who were in the software development field (p1, p3, p4, p6, p8). This became a particularly interesting finding in light of the fact that current headlines in tech indicate that the

software development culture is hostile towards women and their advancement. From these findings, the women graduating from Hackbright and coming into the pipeline are having the opposite experience from many female counterparts in tech, getting support from their male partners in the software development industry. Effectively, these males reinforced the participant's success, motivation, and experience in joining the software engineering industry.

Moreover, the two co-founders of Hackbright Academy were both male. Christian, the instructional co-founder at Hackbright Academy was previously an instructor at Dev Bootcamp. The other co-founder, David, was a student at Dev Bootcamp when they met. Both noted that during their time at Dev Bootcamp they noticed the communication differences between men and women often resulted in women not looking as if they were benefiting as much from the program because they were not getting their questions answered (p16, p17). Since both noticed that "men and women learn differently," they created Hackbright Academy as an alternative bootcamp that would provide more support to female participants' learning. In interviews, both co-founders also noted difficulties they faced while learning computer science. Both are minorities, which may have given them better perspective on difficulties women may face in learning programming (taking into consideration findings from Margolis' study on trends in computer science enrollment) (Margolis and Fisher 2003).

7.1.7 Participants Exhibit Similar Hobbies and Personality Traits

Seven participants described themselves as being introverted (p2, p7, p8, p9, p10, p14, p15). Many baked in their spare time (p2, p3, p13, p14) or liked to engage in arts or photography (p6, p8, p9, p10, p13). Many also engaged in hackathons.

7.2 Interactions at Hackbright

To develop a better understanding of the learning environment at Hackbright, we turn to findings that discuss the social, technical, and pedagogical practices at Hackbright. Specifically, we focus on the social interactions between the teaching staff and peers, the interactions between the peers themselves, and the interactions between the peers and their mentors.

7.2.1 Safe Environment at Hackbright

Four out of 15 respondents mentioned how being in "a safe environment" where everyone could collaborate allowed them to be more "open" to sharing and learning (p1, p2, p14 p15). Four participants mentioned suffering from imposter syndrome (p1, p2, p14, p15), revealing there were many times they felt they were not necessarily the software engineers they were purporting to be. They were constantly worried that they would be "found out."

Additionally some participants described specifically the positive energy of the program and the effect of diversity of ideas. One indicated the interactions "gave her more energy" as there was not a "single day she didn't want to come" (p11). Two participants also noted they would "lose excitement without the community" as diversity "plays an important role in productivity" because "many ideas can solve one problem" and they could not "say which is better than another" (p5, p7).

Many participants also discussed the sense of support they felt, particularly emotional support, and how it helped them cope with and adjust to the pace and challenges of the program (p14, p15, p5). One indicated that the "supportive community" got her where she needed to be (p5). Another indicated the teachers made the experience because she "did hear the instructors were nice" and "if [she] didn't have that….it would be bad" (p2). One participant noted: "you have to be able to ask the questions you don't want to ask, that show you don't know, to be able to understand concepts" (p15). Another indicated she just "really liked the supportive nature of the group" (p2). Many of the participants (5 out of 15) describe themselves as being more introverted and that this environment helped them feel like it was okay to ask more questions when ordinarily they would not feel comfortable doing so (p4, p7, p8, p9, p14).

For one participant, despite describing herself as more introverted, she said she made more of an effort to show her classmates she was supporting them. Moreover the environment allowed her to be more open:

It "reinforces [your] own learning when you have to explain things to people, it made me question if I actually know something, it's just a good feeling to help someone and then people think of you as a resource and I enjoy that role….it's definitely a positive experience." (p14)

This shows the potential power of communal reinforcement: students finding others in the learning environment whom they can mentor or teach can be a significant motivator and detractor from negative feelings they may be experiencing during challenging times in learning. It can reinforce students' knowledge, strengthen confidence, and shift one's focus from personal struggles. Learners instead focus on remembering and restating knowledge to others that so they can continue learning, despite getting stuck or can discuss learning challenges with others. Many participants noted that this supportive environment helped particularly during the intensive learning process (p5, p6). One participant noted that pair programming meant that she "could not give up" on the material she was learning (p6). Even participants who were interviewed after being in the software engineering industry for some time, credited the continued support of their peers, alums, instructors, or "the Hackbright network" as giving them the support they needed to continue to grow in their careers (p6, p15).

Several participants also indicated that the supportive environment "enriched" learning, helped them "feel confident" (p9, p15) and more importantly "helped [them] feel like [they] had a group that believed in [them] (p15). Many participants reported there was a distinction in how they felt (and therefore how they engaged with the material) during

studies in traditional computer science learning and the confidence building environment they found at Hackbright.

Moreover, several participants had taken computer science classes and done online programming tutorials but had experienced an initial learning hump that they could not overcome alone (p2, p5, p8, p10, p11, p14). One participant compared Hackbright to Crossfit, where "you get fit quick" and there's "muscle stress," but you "exert yourself more than you would alone" (p11). Some mentioned coding as something they were afraid of doing initially because of how challenging it was (p7, p9).

One participant noted that it was emotional and hard to get through Hackbright (and "the 48-50 hour work weeks"), "particularly without knowing anything" (p15). She indicated it was "a constant battle" between "feeling like you can grow" and learn, and, feeling like "you couldn't get it" (p15). As a result, she thought a lot of female students did not, and would not, "get" programming at first (p15). In her experience, the initial learning barrier was overcome through:

- "students teach[ing] each other" and
- "instructors spen[ding] a lot of time outside class trying to explain [a concept] in different ways so students get it" (p15).

Teachers effectively became:

- “a shoulder to cry on”, and
- “a catalyst to get students off of being stuck.”

This had the effect of making instructors “more approachable” so that students “c[ould] ask stupid questions” and it was “a safe space to learn” (p15).

Another participant, who had previously graduated with a computer science undergraduate degree, contrasted her undergraduate experience in a traditional computer science program with her experience at Hackbright. She described Hackbright as a safe environment where she “could ask questions multiple times “without pretending she got it when she did not” (p7). In her college computer science classes, by contrast, she felt like “she couldn’t get a word in” and “had to be aggressive” (p7). Moreover, she “felt like she could not be feminine without being judged” (p7). Since “she did not see other people struggle as much as she felt she had,” she reported feeling like “a fraud, and had major imposter syndrome” (p7).

Other interviewees added similar sentiments, noting that the Hackbright community “does a good job” of supporting women, particularly during initial stages (p4, p17). Christian made a point of mentioning that he created a safe space for women before they graduated into an industry with a culture that was not as friendly, “shielding” women from potential “animosity” in industry (p17).

7.2.2 Difference in Age or Temperament

Several participants indicated that age and maturity had a significant effect on how they performed during the Hackbright program and their decision to join Hackbright was based in them being more “aware” of what they wanted because they had matured (p7, p8, p11, p15). One participant specifically added she was not mature in college to make a decision “like this” and it “comes from being older” (p11).

7.2.3 Community Engagement

One participant, after completing the course, co-organized a hackathon event with an instructor at Hackbright (p4). Several additional interviewees discussed participation in community events such as hackathons, often with significant others, peers, or fellow classmates, to boost skill acquisition and awareness of emerging concepts in programming hardware and software applications. Many participants also went onto becoming Hackbright mentors in subsequent sessions.

One participant used her mentors, and mentors of other "Hackbrighters" to network with experts in a particular domain she was trying to become better at, even after failing to get a job after approximately 10 interviews. This participant, who once had a successful consulting career, used her networking skills in "the Hackbright community" to build

relationships that led to job interviews, all while becoming a historian for information learned by authoring a blog. She used the network to give back to others who were also learning and to be a "life coach" to others going through the program. She also used the blog as a vehicle to pick up the industry skills and document knowledge needed to obtain an industry position (she now works as a data scientist in a well-known non-profit).

The social interaction and community building in Hackbright facilitated a lot of interaction that translated into reunions with members of each cohort as well as reunions across different cohorts. Alums used email threads to plan activities including parties, hikes, and craft days. They also used these threads to vocalize new things they had learned and share them with other members of the Hackbright community. That often led to other students wanting to meet, discuss, and learn these new-found tools, applications, etc. in study groups at Hackbright facility (even after graduating).

7.2.4 Many Women Experienced Different Self-Concept Before Hackbright

Several participants noted that they were told by either parents or career counselors that computer science was not a fit for them because they did not have more experience or a math background (p1, p15). Many did not see themselves as programmers before Hackbright (p1, p2, p5, p8, p12, p13). Others reported they took computer science courses and it was disheartening because they could not manage the workload (p11, p14, p15) or find a computer science community at their school to engage with (p2).

For these participants, despite their interest and desire to learn more about computer science, they did not encounter encouragement to keep pursuing their interest in this subject. Despite this, all Hackbright participants who came back to computer science are now working as software engineers, data scientists, or instructors in computer science.

7.3 Interactions with Mentors

Here we switch focus from the social and pedagogical learning environment at Hackbright to discuss the positive *and* negative interactions among Hackbright participants and their industry mentors. Some findings indicated that mentors functioned as peer role models, showing the success possible after Hackbright. Other mentors pushed their students to be better, while some were inaccessible during the mentorship program, despite volunteering for it.

7.3.1 Maximizing the Role of Industry Mentorship in a Bootcamp

Several participants reported a positive interaction with mentors (p2, p3, p5, p6, p8). Many had mentors who pushed them to do better, such as launch an app in the Google Chrome store, pursue a more challenging a hardware project, or provided support by staying with students during debugging (p5, p3, p8, p6). Other participants reported that one significant benefit of encountering alums (and mentors that were alums) was seeing

mentors who were younger and very successful after Hackbright (p1, p2, p3, p6, p7, p10, p11, p13, p15).

Hackbright Academy's mentorship program, which is mostly male, became a test-bed in how to facilitate better mentorship interactions between female students and male industry professionals. Through a series of trial and error approaches to improving interactions between mentors and mentees, Christian indicated that problems with communication or learning styles became much less prevalent when there was an initial training session for mentors during each 10 week session (p17). The mentor training functioned as a type of communication class and sensitivity training in how to effectively communicate while teaching, particularly to this population of female students.

7.3.2 Is it Better to Have Mentorship or Support from a Personal Network of Friends and Significant Others?

Several participants (p14, p2) mentioned that having up to three mentors was challenging because it was hard to "juggle" those mentors with their workload. Christian said that often mentors would object to what could be accomplished during the 10-week program. Some mentors said that participants' projects were impossible in the time allotted or that participants were not qualified for hire in comparison to those who had more exposure than the 10-week period. Other participants mentioned being disappointed by mentors who remained largely inaccessible while they were struggling through their projects.

Moreover, several participants mentioned that when they did receive help from mentors, it wasn't always helpful because it was too "granular," delving into very specific things, while participants were still trying to get a grasp on how to get their code to work properly (p2, p14). These participants instead reached out to friends, or their significant others, who were more accessible through instant messaging, text, or some other social media forum or in person. The reason for this was that friends or significant others "were more straightforward" or accessible (p1, p14). They would "just indicate a specific library needed to be used in a project and here's the link," whereas a mentor was "too low-level" and "more theoretical", instead of just being practical (p14).

7.3.3 Challenges with the Industry Mentorship Program

Some participants reported having mixed feelings about mentors because it was hard to arrange a visit and there was no guarantee that the help would help them progress with their projects (p1, p3, p4). Additionally, some mentioned they didn't feel they could be completely honest with their mentors, indicating that they felt that "they can't be vulnerable" because their mentor was "weary of accelerated bootcamps" or because they had "asked twice and fe[lt] bad" (p1, p2). Others reported not using their mentors because of differences in personality (p3, p8). Some did not end up using mentors to ask about projects or programming questions at all (p2, p11).

7.3.4 Recommendations for Mentorship Program

Some participants (p2, p14) mentioned having mentors that were not knowledgeable in an area related to their project. They also had difficulty in coordinating schedules. Suggestions for improvement included having fewer mentors and ones who had skills that were more in line with a participant's' project. Some participants (p2, p14) mentioned feeling obliged to reach out even when so many mentors were needed. One participant thought it would be helpful to rotate mentors (p14). She wished there were a tool where mentors were "forced" to hang out with their mentees to get to know them better, not just working, but also bounding, in order to create relationships where they could continue to get feedback throughout the program.

Christian indicated a key element of having better interactions between mentors and mentees was establishing a protocol for interactions with the mentors. This would consist of Hackbright having a mentor-training session providing information on how to constructively provide feedback to participants so they could improve and stay motivated despite learning challenges.

7.4 Challenges in Hackbright

7.4.1 Downside of Taking on More Challenging Projects

The individual-project periods (the last five weeks of the course) are the points in the program where students have more one-on-one time with instructors. One student described it as a critical time to develop relationships with instructors (p14). For those students who chose to take on material that instructors are not familiar with, this time period becomes isolating since those students cannot ask for sufficient help from instructors (or other students). These participants do not experience the benefits of communal support because no one in the cohort (or on staff) is familiar with the elements of their project. One participant said her alternative was to instead engage in "rubber ducking" (a mode of self-help where one talks to a rubber duck to try to talk out the answer to a question posed), which was very isolating (p14). She noted that bonding forms when one is getting help from instructors because they become invested in the projects/artifacts of students they are helping. Instructors will give high fives when their students get a concept that is important to their artifact, but for the participant taking on more challenging material, her victories are not noticed. The more instructors interact with a student and help her build an artifact, the more invested instructors become in the participant's progress and success. If teaching staff help students with coding, it's similar to the participant having another mentor, whom they can bond with. One can argue that bonding helps creates the safe environment for continued learning mentioned throughout

participant interviews, which propels students to continue working through difficult parts of their projects.

Additionally, this participant had difficulty in pair programming exercises because she got paired with partners who "did not know as much and had different beliefs on how something should work" (p14). Hence she preferred to work alone and at her own pace. Her experience was different from her peers as she was learning more challenging technology than her peers (Objective C) without communal support or encouragement from her peers or teaching staff.

7.4.2 Difficulty Understanding All the Concepts in Short Amount of Time

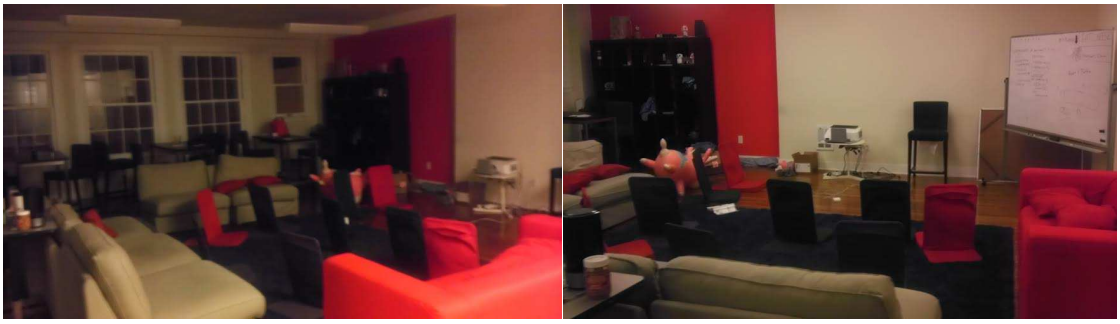
Another participant (p15) stated it was hard to understand concepts, stating "it took me a lot more time to write code for problems." The short duration of the program (10 weeks) was seen as not sufficient to obtain a strong grasp of the material. Hence she (and several other) participants spend months after the program studying for coding interviews until they are comfortable enough with the material to do well in a "whiteboarding" interview. One participant had to go back and practice after finishing Hackbright by taking online courses such as MIT's CS 21 Python course and EdX's 6.0 Introduction to CS. Although this participant went through 10 initial interviews without getting an offer, with her first one being at a partner company, Facebook, she is now working as a data scientist at Change.org.

7.5 Suggestions for Improving Other Constructionist Spaces

7.5.1 Recommendations for Use of Space in Constructionist Environment

As part of our research findings, we present recommendations on the intelligent design of an informal constructionist space. We present a collaborative design model based on interview findings from Christian during his instruction at Hackbright.

With each cohort (he was lead instructor for 5 cohort sessions), Christian redesigned and adjusted aspects of the space to facilitate more learning and fewer distractions.



Figures 7 & 8: Hackbright "learning space"

Christian advocated being cognizant of reactionary learning differences based on usage of space and furniture. He suggested designing a space, shown in Figures 7 and 8, that facilitates comfort and community (couches and "backjacks" enable communal attentiveness, while coffee tables enable zoning out). He indicated that having a shared

space where the learning area is next to the kitchen is not incidental. He specifically wanted students to be there "hanging out," and even welcomed student-planned sleepovers as they indicated continued interest in doing the work.

Another consideration is for design of a constructionist space is class size and its marked effect on the overall community building of a cohort. A smaller class size is recommended to maintain a communal aspect to the students in the class. Christian commented that when he had 30 students, the communal aspect of the classes faded, but when the classes increased to 40, the result was a more fragmented and divisive labspace, instead of collective one. At a class size of 40, students formed cliques of 2-4 people instead of a singular community or even 2 smaller communities.

Another recommendation made was to create a motivational area in the space where learners can be expressive and encouraging to each other. At Hackbright, these spaces included a motivational bathroom and an area displaying photos and plaques. Creating a space that is motivational allowed participants to feel more at ease using nearby resources during challenging times (i.e. students had sleepovers at the facility and oftentimes came back "to hang out" since each cohort member kept their keys even after graduating).

Lastly, another recommendation made was to develop an awareness of how the activity level (and positioning) of instructors affects participant's willingness to ask questions. During sessions, Christian mentioned that students would hesitate to get up to ask

questions if they saw teaching staff sitting or communicating with each other instead of focusing on being available for nearby students. Hence, he made a point of remaining active physically during class sessions to stimulate continued confidence in students asking questions.

7.6 Constructionist Findings

Our literature review of constructionist environments discussed several types of constructionist environments and a shift from initial tinker based masculine constructionist environments (LogoBlocks, Mindstorms, construction kits) to more social ones that were virtual but non-visual (MOOSE Crossing, MediaMoo), to constructionist environments that were visual both in the virtual and physical spaces (Alice and Resnick's Computer Clubhouse, respectively). We then saw a shift from constructionist activities as hobbies (refer to Figure 6, Phases 1, 2, 3) to activities that took place in schools. These activities (Phase 4 in Figure 6) include designing virtual mobile apps in Google App Inventor and "making" things in school-run Fablabs. Lastly, we see the professionalization of constructionist activities (Phase 5 in Figure 6) where professional products are created for sale and distributed in hardware startups, incubators, and makerspaces. Additionally, constructionist activities are used to professionalize *people* (via software bootcamps) through artifact creation so that they can become software engineers.

We now shift our focus to discussing instances where Hackbright participants created artifacts that were demonstrative of their learning for professionalization. We pay special attention to instances of social constructionism and distributed communal support.

7.6.1 Examples of Constructionism

Hackbright participants engaged in constructionist activities that were external and shared during pair programming exercises and individual projects. In both activities, they acted as central agents in their learning. For their individual projects however, Hackbright participants developed individualized and personally meaningful artifacts that they shared not only with instructors and peers for feedback, but also with an extended network that included industry mentors (each Hackbright student got 1-3 mentors), Hackbright alums, and other technology professionals that chose to be part of the Hackbright network by coming to networking dinners, industry events, and student showcases.

The surrounding culture of Hackbright teachers, industry mentors, Hackbright graduates, industry sponsors, tech speakers, industry mentors, and fellow peers all became resources for Hackbright participants to build and extend their knowledge in the short time they were given. These resources also allowed them to take what they learned, refine it, present it, measure its effectiveness by obtaining feedback in all these channels, if not support from these surrounding cultures. They could then reiterate and present their construction of those concepts in an individualized way.

Hackbright students also discussed and shared their constructions throughout the course during pair programming exercises (during the first five weeks of the course) and while creating their individual projects (during the last five weeks of the course). Individual projects included mobile apps, foreign language programs, and desktop games.

Much of the constructionist discussions were localized, with students often consulting each other, or one of the five instructors, for feedback and guidance. Other discussions, those that involved distributed constructionist activities, often took place by students sharing their constructions with friends, Hackbright alums, or more prominently, industry mentors. Each Hackbright student was assigned 2-3 industry mentors to help them develop their projects and prepare for a career day in which they would showcase their projects to approximately 25 partner companies,

The distributed network that students at Hackbright referenced included students, alums, previous mentors and/or affiliates of Hackbright. Much of the discussions and sharing took place through this distributed network at events designed to foster development of relationships, such as industry events, tech talks, networking dinners, and receptions (i.e. Girl Geek Dinners). There are also student demo showcases for students to share their work with the public. Sharing their work through this venues enabled students to reiterate ideas and concepts tied to their projects so they could create an artifact that was relevant and up-to-date with current technologies in industry.

One student interviewed indicated she blogged about her learnings, using the blog as a forum to discuss, share, and further reiterate on development projects she was taking part in. She also consulted those in the distributed Hackbright network to help her take on projects in a technical area she was less familiar with so she could improve herself before going onto the job market.

7.6.2 Social Constructionism

In our study, social setting played a significant role in artifact creation. The community of Hackbright participants encouraged taking on identities as software engineers through activities such as receiving business cards (as shown in Figure 7) so they could internalize and externalize their role as software developers early in their training. Their business cards were linked with their artifact creation (i.e. Gitub accounts that facilitated development and sharing of their online artifacts as shown in Figure 9). This allowed for participants to more easily assimilate into a software development role, but also allowed for artifact creation to be prominently linked to their title (through Github repositories).

Figure 9: Hackbright Participant's Business Card

In addition, while Hackbright's social setting included traditional lecture-based daily activities for the first five weeks, emphasis was also given to constructionist activities like pair programming exercises where participants could choose different partners every day. Hence, the social setting that stemmed from these activities promoted relationships where a usually-marginalized demographic (women in computer science) could develop strong bonds, provide emotional support, all while building a network that would support their own individual and shared artifacts in that setting.

The social setting at Hackbright also included going on tours of companies like Github and Google, while holding "tech talks" featuring industry speakers from a specialized software development area (Figure 10 shows an example of the logo used at a recently co-sponsored

dinner with Hackbright and industry partners). Access to these industry professionals, particularly ones discussing and sharing their craft, allowed Hackbright participants to more critically understand facets of the environment they were trying to become a part of. In addition, by attending tours at tech companies like Google, Github, Square, etc., they were able to build informal relationships with people representing those companies, and use those relationships to not only be perceived and addressed as software engineers, but to practice assimilating into the field by discussing their Hackbright experiences and projects.

Figure 10: Logo for Co-Sponsored Industry Related Hackbright Dinner & Reception

Lastly, the network of Hackbright alums, industry partners, mentors, etc. created a setting where the network (and people in it) became a vital resource for knowledge, networking, job recommendations, mentoring, etc. This enabled students to improve and refine ideas and skills used towards creation of artifacts. In addition, many interviewed shared a common vision to change the world through beginning to engage in dialogue that was once

restricted solely to software developers creating technical products, but also through changing a predominantly male status quo in the software development industry.

Moreover, in interviewing students, it became evident that Hackbright participants did not just construct their projects or knowledge, or engage in various forms of distributed/social constructionism, but that their constructionism took new forms.

Hackbright participants did not just construct these objects or relationships that facilitated learning, they constructed coping mechanisms to deal with challenges in learning. For example, on a micro-level, this constructed coping mechanism to learning challenges took the form of an anonymized wall of motivational quotes in one of the female bathrooms. This social practice started during the third or fourth cohort before becoming a part of the community support in Hackbright. What is especially noteworthy is some participants would take pictures of the wall and look at it during particular moments when they needed encouragement. In this sense, participants constructed anonymized support to help other students when they were at their absolute lowest.

On a macro-level, the support system, which was really propagated through a shared email thread provided by the co-founders of Hackbright, and social events run by Hackbright, enabled the creation of a sub-society of engineer graduates who could not only provide emotional, intellectual, and professional support, but could also continue to further social and career-oriented pursuits even after graduating the course. Students' constructions

therefore, were not just physical individualized projects, but also cultural and social systems to facilitate continued existence, if not success, in the computing workplace.

7.6.3 Types of Distributed Communal Support

Participants keep in touch through emails and in person after the program. A co-founder indicated that Hackbright provided a mailing thread such as those used in Y Combinator to facilitate communication among participants. The email thread was then used to facilitate further get-togethers (parties, reunions amongst cohorts, study groups for new material to learn even after completing the program). Several participants (p14, p15, p6) indicated there were "a bunch of threads" and a number of people who go on there to share technical information or vent about difficulties in workplaces. Many participants communicated with each other even after graduation almost every day (p14, p15, p6).

7.7 *Computer Supported Cooperative Learning*

At Hackbright, over the shoulder learning was used in helping participants develop an understanding of programming environments, languages, and to help students in building hardware related projects (p5).

We found that participant interactions with other students and industry mentors, particularly during final projects and technology talks, allowed for continued interest and

both over-the-shoulder learning and apprentice learning, both in collocated and distributed spaces. Pair programming exercises also allowed for incidental learning and over-the-shoulder learning.

7.8 Pair programming

7.8.1 Benefits of Pair Programming

One student labeled pair programming as a tool for self-improvement because students had to improve themselves with communication issues (i.e. not being too forward in solving a problem, particularly if it wasn't their turn). Additional discussion focused on the effects of interaction during the teamwork in pair programming. For example, if a student did volunteer an answer out of turn, one mentioned feeling "dumb" because of it, since pair programming lab exercises are not just about arriving at the solution, but also about teamwork. More specifically, she stated she learned that as a partner she had to accommodate her role in the pair programming interaction (i.e. be respectful of who's turn was it or who got "there first") and work within that paradigm to facilitate a collaborative solution.

Additionally, other students reported that the pair programming process enabled them to learn how to talk through problems and get comfortable vocalizing how code works (which turned out to be a key skill for job preparation). One student reported that pair

programming, while partners worked with someone else, allowed for a leveling out of the pace of the problem solving (p1).

Another student indicated she learned the pair programming process was not just about the solution, but also about *getting feedback* during the process.

7.8.2 Challenges of Pair Programming

Another student discussed hidden challenges in the navigator role. While the student leading the pair programming (navigator) could think big picture, the partner (the driver) could get caught up in tiny details and encounter challenges. The challenges would stem from both listening to the navigator discuss her proposed solution *while* considering (*internally*) the details of that solution and potential debugging issues (p8). Hence it took a certain amount of *being focused* in order to handle the navigator role, even though it was considered to be more of a "backseat" to the driver who directly conveys a direction for implementing a solution. Another student indicated that while a driver could take over the entire process/program, a navigator had to think more, pick out mistakes, immerse herself and look at code, and also could encounter difficulties by getting lost in a train of thought (p7). While some students reported that pair programming made them feel less pressure in coming up with a solution (p5), others indicated they were harsh towards themselves during paired exercises (p1). One student indicated that if she was working alone she

would take responsibility for problems she solved, but in a pair she would give credit to her partner for good things and blame herself for the bad things (p1).

Many students indicated that pair programming was very taxing because they had to *constantly* interact and engage with a partner (p15, p10). Some indicated they would counteract this pressure by taking more breaks. One student indicated she felt safer as a navigator and more "put on the spot" as a driver to solve problems because she usually wanted whatever time was needed to solve the problem (p5).

Another indicated that working with another person resulted in taking double or triple the time to submit a solution, so partners had to learn time management (p5).

7.8.3 Recommendations for Improving Pair Programming Practices

Some interviewees made suggestions for things to consider during the pair programming process. One specified the key to the process was the understanding that *patience and communication* are the key skills to making a pairing successful (p10). Another student made recommended that students take a communication class (p15).

Moreover, one student reported that during pair programming, it was really important to make sure not to discount a partner's capabilities if they lacked knowledge in certain subjects out of the gate as it's possible to learn from a partner who is learning if there is

receptivity to the idea (p15). On the flip side, a pair programming partner who has "less knowledge" should not "shut down" if she was uncertain about concepts initially and push on with the questions because the space is for learning and mistakes are okay (p15).

Another student indicated that it took time to develop a good working rapport to be in sync. Specifically, even though promiscuous (rotating) pairs could be put together immediately, work ethics might be different, with some not talking as much as others, or others talking more thoroughly (p7). Hence working on developing a good rapport could offset differences in work ethics or communication styles if pair programming partners worked on improving these skills. According to one student, promiscuous pairs were beneficial because the student knew the following day would be "totally different" and she would have an opportunity to try a new approach, in optimizing her learning or acquiring a skill (i.e. "maybe tomorrow my partner will be really good at visualizing the problem and I'll be really good at knowing the specific methods and classes we need") (p5).

7.9 Collocation Practices at Hackbright

At Hackbright, Covi's finding that collaborative practices should include a shared display and awareness of team members' activities is evident but through more modern practices such as pair programming and scrum meetings. Shared spaces for learning and building at Hackbright facilitated not only improved productivity and sharing of tips to increase efficacy, but led to a more communal environment that encouraged communication, unity,

and an increased willingness for individuals to step outside their own personal comfort zones to become better for their Hackbright classmates/teachers.

With Teasley's work in mind, we see the benefits of radical collocation on the productivity of students. Christian noted that mentors often made sure they were collocated with their mentees to prevent communication issues when their mentees were seeking help with personal projects. Surprisingly, one of the benefits of collocation in mentor/mentee relationships was the *promise* or *potential* productivity that could occur as a result of being collocated in the same space, even if that access to that *promise* or *potential* help was not realized (p17). Collocated mentors could also assist Hackbright interviewees with career development such as whiteboarding (solving computer science problems on a whiteboard to share ideas while solving) or could boost morale, motivation, and productivity to their mentees (p17). Hackbright participants would often meet mentors at coffee shops or the mentor's workplace to acquire necessary information for contextualizing their projects in existing industry practices.

At Hackbright, many components of the program replicate or prepare students for workplace practices in technology companies like the one studied in Teasley's research. Hackbright students work in paired teams often, having to explain their direction, progress, and approach during lab exercises. Many companies today still use pair programming exercises in the workplace to increase productivity and reduce error rates in code. Radical collocation was also effective during pair programming because it allowed partners in

teams to learn “tricks” from their partners during programming. These tricks included ways to simplify their development environment or shortcuts to help during programming (p3, p6, p8). Radical collocation also allowed for increased productivity while participants worked on their individual projects because they could use each other’s increasing knowledge to resolve problems they were encountering with new vocational technologies (p8).

Participants during their individualized projects worked in one collaborative lab space to help each other through challenges in their individual learning. This dedication to using a specified space for constructionist activity and vocational training allowed for increased motivation (p2 noted that during the most challenging times of her project, many of her Hackbright peers and she would sit together and high five each other if they moved forward past a roadblock). Additionally, Christian indicated that he preferred smaller class sizes (around 20 or so) to enable a more cohesive group that could learn from each other’s perspectives without becoming so big that it could become divided.

Additionally, advantages such as communal support in a collocated space were evident during scrum meetings. Participants would report on their own individual progress with their projects. The benefits of this were two-fold:

1) scrum meetings enable on the job training

2) allowed for participants to track their progress, while using the collective intelligence and expertise of teaching staff and peers to work through problems they were facing.

Chapter 8- Discussion

In the following section, we provide an overview of the results of our study on Hackbright and its graduates. We discuss contributions our study makes to existing literature on constructionism, collocation, computer supported cooperative learning, and pair programming. We also discuss implications for design for bootcamps at large, and Hackbright specifically. Lastly, we discuss future directions for our work.

8.1 Summary of Results

The findings of our study revealed numerous things. Many women who joined Hackbright did so for financial security (5 out 15), out of a desire to find a fulfilling job (7 out 15), due to lack of mobility/disempowerment in their previous line of work (8 out 15). Many had previous exposure to technology through work or school (6 out 15). Several had the support of friends or family in the software industry (5 out 15). Many also had partners in the software industry who supported them during their fellowship (5 out 15). Moreover, most students knew someone who had gone through the Hackbright program (9 out 15). That enabled them to get a sense of what going through Hackbright and getting a software engineering position would look like. The "role models" of these Hackbright graduates were not older or established in software engineering, they were peers, who stepped up and transitioned into software engineering just like they had. Their example made it possible to envision taking that step.

Preliminary demographics also revealed that many of the interview participants, were previously teachers, camp counselors, or coaches (8 out of 15). Additionally some women interviewed reported that the "open" environment present in the software engineering fellowship facilitated becoming more extroverted in classroom activities (4 out of 15).

Fifteen out of 15 participants (all the interviewees) had successfully transitioned into software industry positions after a one year follow-up. One was teaching at Hackbright, another was a data scientist, and the remaining 13 had secured software engineering roles in various companies like Keen.IO, Crittercism, and SurveyMonkey.

Two out 15 reported that their older age enabled them to be more mature when deciding to join Hackbright as they were not mature enough to make this kind of decision during college (and even went through grad school not knowing how to ask questions).

Ten out 15 participants reported having a very positive experience at Hackbright in terms of both learning and social interactions. They indicated the instructors were approachable, allowed students to ask questions as many times as needed, and would often try explaining concepts multiple times in an effort to make sure students understood the concepts before going on. This type of support enabled students to engage in the "confidence building" necessary to get past initial learning humps that they could not overcome alone. Two

reported that they felt their age and maturity played a role in their decision to come to Hackbright as they were more “aware” and were not ready during college to make big decisions regarding their career.

Findings on the mentorship program indicated there were both pros and cons for mentees. One pro included the students getting a realistic depiction of what it would be like to be a software engineer after Hackbright (2 out 15). In fact many reported that it was inspiring seeing a Hackbright alum who was very successful after Hackbright and found the work fulfilling (9 out 15). Many mentors provided mentees with help for broad concepts, including interviewing, but not technical help on projects (3 out of 15). Other pros included mentors helping with professional development and encouraging students to be better or helping with networking opportunities (3 out of 15). Some mentees also reported difficulties with having mentors because they did not feel safe in asking them questions (i.e. asked twice and felt bad asking again or did not ask much because mentor indicated he was weary of bootcamps in general; 2 out 15). Others indicated they did not engage with their mentors because of personality differences so the relationship fizzled (2 out 15).

8.2 Challenges and Considerations for Hackbright

Since a few participants did experience difficulties with Hackbright, one must look at possible ways to circumvent these difficulties in the future. One student indicated she had significant difficulty because the course was only 10 weeks making it hard for her to really learn the concepts (p15). Another struggled with the social environment because she was more advanced than her peers, leading to isolation during activities such as pair programming and project time, where she ended up working alone. She reported feeling isolated because it was not easy to get help and she was not congratulated for her learning milestones the same way her peers were.

This gives room for thought on creating software development bootcamps that might last longer (approximately 15 weeks) to give students more time to learn the material adequately. Another possible solution to accommodating a more advanced student is to make sure one or more of the teaching staff is knowledgeable enough to accommodate student's skill level in the class prior to the start of the fellowship so they do not become isolated.

8.3 Mentorship Program Recommendations

We add to the existing literature on constructionist environments by discussing the benefits and disadvantages of mentorship in these professionalized software development

bootcamps. We note the following recommendations made by students regarding having a mentorship program in a bootcamp:

- having mentors paired with students they can build a bond with
- having mentors who had skills that were more in line with a participants' project
- rotating mentors so participants could have more access to learning new information
- create mentorships where mentees can build relationships where they could continue to get feedback as their careers progress.

8.4 Pair Programming Recommendations

Our work adds to existing literature on pair programming. Participants reported on difficulties, benefits, and suggestions to improve the experience for other students in the future, regardless of whether they were learning in an informal or formal learning environment.

Previous literature in this field discussed mostly quantitative findings on solo vs. pair programming scores on assignments and finals (McDowell, Hanks, and Werner 2003; McDowell et al. 2002; McDowell et al. 2003) or how students explain the benefits of rotating pairs (L. Williams et al. 2000), or discuss whether the quality of a computer science class would improve with pair programming (McDowell, Hanks, and Werner 2003). No studies have discussed pair programming in bootcamps and none have given qualitative

feedback for non-technical women on ways to improve pair programming practices. With this in mind, we share the following pair programming recommendations made by students.

Suggestions to improve pair programming in the future include:

- Understand that patience and communication are the key skills to a successful pairing
- Take a communication class prior to working in pairs
- Do not discount a partner's capabilities if they lacked knowledge in certain subjects as it is possible to learn from a partner who is [still] learning.
- Do not shut down as a pair programming partner who has "less knowledge" and uncertain about concepts initially. Just push on with the questions because the space is for learning.
- Develop a good working rapport to be in sync as it can be a vehicle to offset differences in working styles, communication styles, and work ethics.

8.5 Considerations for Constructionist Activities

We identify and discuss engagement in constructionist activities in this new type of constructionist space that teaches not only the technical skill but professional practice as well. Students discussed and shared constructions in pair programming, during shared lab space time, and during interactions with their mentors (in person and online). They

engaged in distributed constructionist activity by collaborating with mentors or peers through Github, Skype, or messaging applications. They also engaged in constructionist activity through setting up meetings for professional development, discussing jobs, and raising questions through an email list serve linking all Hackbright graduates.

Two particularly unexpected but important forms of social constructionism occurred among Hackbright graduates. First, they began to organize cross-cohort functions including parties, reunions, and get-togethers, using the email list serve. The effect is continued support, networking, professional and social development that encourages growth and self-sustained development among graduates in the group. Secondly, graduates created an anonymized wall of "motivational quotes" to inspire each other during the peak of their learning challenges. In effect, their constructions were not just the artifacts, but the *coping mechanisms* used to break through learning challenges.

Our contribution is to identify these new and unique ways Hackbright graduates constructed social networks and coping mechanisms to ensure survival and getting through difficulties during the fellowship.

8.6 Considerations for Collocation Practices & Computer Supported Cooperative Learning

We identified collocation practices in this new setting, software development bootcamps for post-collegiate women. Specifically we identify instances during mentor and mentee interactions where radical collocation enabled better efficacy in artifact construction for personalized projects through computer supported cooperative learning. Radical collocation led to improved efficacy during pair programming and artifact construction since peers could share knowledge they had gained in order to improve both collaborative and individual constructionist activities. Over-the-shoulder learning and incremental learning also occurred among between mentors, teaching staff, and mentees during teaching sessions. Mentees often learned tricks to simplify designing their projects or setting up their development environment. Radical collocation also allowed for better professional development as mentees could practice interview questions with mentees by whiteboarding.

8.7 Implications for the Design of Bootcamps

In the following section we discussion the implications of our data, specifically that all 15 participants interviewed were able to secure a job in software engineering or data science after taking the course. We look at the implications of bootcamps and their success in transitioning non-technical adults into software engineering. We also look at the

implications of Hackbright's success as a bootcamp, discussing significant considerations in our data (and potential future work).

8.7.1 Effective Transition for Women into Software Engineering

On a macro-level, there is an important reason why software engineering bootcamps, and specifically Hackbright Academy, have been able to place and keep its graduates in software engineering roles. According to Christian, there's a hidden secret in tech--a distinction between vocational skills and theoretical skills in software development. The two do not necessarily overlap. Even if one candidate had all of the desired software development training a company was looking for, they would *still* not be of value to them for the first 1-6 months. Hence, once a Hackbright graduate gets hired into a software engineering role, even if she was not as skilled as a college computer science graduate, she was given time to adjust to the new. What becomes crucial, according to Christian, is the new-found confidence the students acquire. It's more paramount than skill in interviews. Hackbright facilitated establishing confidence in several ways including giving participants software engineering business cards with their Github accounts listed. According to him, "they have to believe it so they can sell it."

There is also another reason why bootcamp candidates might have an advantage over traditional computer science graduates. Graduates from bootcamps are trained in the vocational technologies used by partner companies so their knowledge in grounded in

current industry practices (Kamenetz 2015). In contrast, students at traditional four year schools are taught more theoretical languages such as Java and C++, which may or may not be appropriate for the company making the hire. Furthermore, after the first year or two in traditional computer science majors, there becomes such variation in the courses offered to graduates that what is being taught is not necessarily what a student will want or need to learn in order to obtain a desired job. Classes are chosen subjectively and are not necessarily fundamental to obtaining a software engineering position. Additionally, getting into the bootcamp program is competitive. Surviving it demonstrates a level of work ethic that could be comparable to, if not superior, to that of a graduate from a traditional four year university.

On a micro-level. Hackbright is distinctive from other bootcamps because of its selection process. Many bootcamps recruit their students by having them attempt or complete a series of programming challenges before deciding whether or not they are accepted into a bootcamp session. Hackbright does something radically different. In a phone interview with potential candidates, they are to talk about a skill they picked up and break down every relevant and notable component of that skill and teach it to the interviewer. In essence, the phone interviewer is trying to ascertain the candidate's ability to break down information presented (the same way one would do when writing a program) and to teach the information they have become experts at (which makes coding the computer analogous to teaching). The more clear, concise, articulate the candidate is in relaying that information, the more desirable they become as a candidate. For Hackbright it's not about

previous exposure in computer science, but about testing a candidate's ability to communicate (sometimes complex) ideas succinctly because that is essentially what programming, as a skill, entails.

One implication from this study can be to look at the kinds of learning environments that facilitate students continuing to exhibit confidence in challenging material such as software engineering. Hackbright enabled female students to have confidence by consistently providing a safe space for them to learn while getting emotional support, encouragement, and unrestricted access to a space where they could meet. The result is all the participants interviewed are working in software development as engineers, data scientists, or software development instructors (and have been working in it for over one year).

8.8 Future Directions

One surprise in this research was the mentorship training (to mostly male mentors) given by the instructional co-founder in how to address software learning challenges, while acknowledging specific communication and learning differences to women (Tannen 1991). There are many trained professional women leaving the software industry because of the "hostile" culture. This training could facilitate mentors' communication, sensitivity to learning styles, and awareness of what women can accomplish in such a small timeframe. This in turn could create working environments to be more women-friendly. This is an important researchable question.

More importantly, the success in training mentors to have smoother interactions with their mentees has significant implications for design for software engineering bootcamps and mentorship programs in tech at large. If the training that Hackbright gave to mostly male mentors became effective in cultivating better working relationships between men and women in software engineering through mentorship, what is not to say that it might be an effective tool in addressing these differences in the software industry in general? With many accomplished and professional women leaving the tech industry today because of the unfavorable culture, training that could facilitate better communication might be a necessary component in changing the prevailing culture in tech.

In future research, we hope to conduct further studies on defining and understanding the mentorship training used in the bootcamp to facilitate smoother interaction between men and women in the space. We then hope to determine the potential effectiveness of a model that could effective to bridge difficulties in communication and learning in the software engineering industry.

BIBLIOGRAPHY

AAUW. 2000. "Educating Girls in the New Computer Age." *American Association of University Women Educational Foundation, Washington, DC, USA*.

Ackermann, Edith. 2001. "Piaget's Constructivism, Papert's Constructionism: What's the Difference." *Future of Learning Group Publication* 5 (3): 438.

Berlin, Lucy M., and Robin Jeffries. 1992. "Consultants and Apprentices: Observations about Learning and Collaborative Problem Solving." In *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, 130–37. ACM. http://dl.acm.org/citation.cfm?id=143471.

Bevan, Jennifer, Linda Werner, and Charlie McDowell. 2002. "Guidelines for the Use of Pair Programming in a Freshman Programming Class." In *Software Engineering Education and Training, 2002.(CSEE&T 2002). Proceedings. 15th Conference on*, 100–107. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=995202.

Blikstein, Paulo. 2013. "Digital Fabrication and 'Making'in Education: The Democratization of Invention." *FabLabs: Of Machines, Makers and Inventors*, 1–21.

Boyer, Kristy Elizabeth, James Lester, Bradford Mott, and Eric Wiebe. 2014. "Toward a Computer Science Learning Progression: Investigating the Role of Adaptive Learning Environments for K–12." Accessed March 14. http://www.stanford.edu/~coopers/2013Summit/BoyerKristyNCSU.pdf.

Bruckman, Amy. 1998. "Community Support for Constructionist Learning." *Computer Supported Coop. Work* 7 (1-2): 47–86. doi:10.1023/A:1008684120893.

Bruckman, Amy, Maureen Biggers, Barbara Ericson, Tom McKlin, Jill Dimond, Betsy DiSalvo, Mike Hewner, Lijun Ni, and Sarita Yardi. 2009. "Georgia Computes!: Improving the Computing Education Pipeline." In *ACM SIGCSE Bulletin*, 41:86–90. ACM. http://dl.acm.org/citation.cfm?id=1508899.

Carver, Jeffrey C., Lisa Henderson, Lulu He, Julia Hodges, and Donna Reese. 2007. "Increased Retention of Early Computer Science and Software Engineering Students Using Pair Programming." In *Software Engineering Education & Training, 2007.*

CSEET'07. 20th Conference on, 115–22. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4271597.

Cliburn, Daniel C. 2003. "Experiences with Pair Programming at a Small College." *Journal of Computing Sciences in Colleges* 19 (1): 20–29.

Cockburn, Alistair, and Laurie Williams. 2000. "The Costs and Benefits of Pair Programming." *Extreme Programming Examined*, 223–47.

Covi, Lisa M., Judith S. Olson, Elena Rocco, William J. Miller, and Paul Allie. 1998. "A Room of Your Own: What Do We Learn about Support of Teamwork from Assessing Teams in Dedicated Project Rooms?" In *Cooperative Buildings: Integrating Information, Organization, and Architecture*, 53–65. Springer. http://link.springer.com/chapter/10.1007/3-540-69706-3_7.

Dann, Wanda, and Stephen Cooper. 2009. "Education Alice 3: Concrete to Abstract." *Communications of the ACM* 52 (8): 27–29.

Dewey, John. 2007. *Experience and Education*. Simon and Schuster. https://books.google.com/books?hl=en&lr=&id=JhjPK4FKpCcC&oi=fnd&pg=PA14&dq=dewey+experience+education&ots=D9uHZsGCIf&sig=OuuXNCyJY5jJ-EsiiGo9oHbDSt8.

Forte, Andrea, and Mark Guzdial. 2005. "Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses." *Education, IEEE Transactions on* 48 (2): 248–53.

Hackbright Academy. 2015. "Hackbright About Me." *Hackbright Academy*. Accessed March 13. http://hackbrightacademy.com/about/.

Hartness, Ken T. N. 2011. "Working to Change Perceptions." *J. Comput. Sci. Coll.* 26 (5): 259–67.

Kamenetz, Anya. 2015. "12 Weeks To A 6-Figure Job." *NPR.org*. Accessed January 14. http://www.npr.org/blogs/ed/2014/12/20/370954988/twelve-weeks-to-a-six-figure-job.

Karakus, Murat, Suleyman Uludag, Evrim Guler, Stephen W. Turner, and Ahmet Ugur. 2012. "Teaching Computing and Programming Fundamentals via App Inventor for Android." In *Information Technology Based Higher Education and Training (ITHET), 2012 International Conference on*, 1–8. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6246020.

Kelleher, Caitlin, and Randy Pausch. 2005. "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers." *ACM Computing Surveys (CSUR)* 37 (2): 83–137.

Klawe, M. 2013. "Increasing Female Participation in Computing: The Harvey Mudd College Story." *Computer* 46 (3): 56–58. doi:10.1109/MC.2013.4.

Kolko, Beth, Alexis Hope, Brook Sattler, Kate MacCorkle, and Behzod Sirjani. 2012. "Hackademia: Building Functional rather than Accredited Engineers." In *Proceedings of the 12th Participatory Design Conference: Research Papers-Volume 1*, 129–38. ACM. http://dl.acm.org/citation.cfm?id=2347654.

Kuznetsov, Stacey, Laura C. Trutoiu, Casey Kute, Iris Howley, Eric Paulos, and Dan Siewiorek. 2011. "Breaking Boundaries: Strategies for Mentoring Through Textile Computing Workshops." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2957–66. CHI '11. New York, NY, USA: ACM. doi:10.1145/1978942.1979380.

Lave, Jean, and Etienne Wenger. 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge university press. https://books.google.com/books?hl=en&lr=&id=CAVIOrW3vYAC&oi=fnd&pg=PA11&dq=lave+wengar+situated+learning&ots=OBmErpXIEl&sig=NP91LuhnB63YdeVUZ4PJgZ7MTHs.

Lindtner, Silvia, Garnet D. Hertz, and Paul Dourish. 2014. "Emerging Sites of HCI Innovation: Hackerspaces, Hardware Startups & Incubators." In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, 439–48. ACM. http://dl.acm.org/citation.cfm?id=2557132.

Marcu, Gabriela, Samuel J. Kaufman, Jaihee Kate Lee, Rebecca W. Black, Paul Dourish, Gillian R. Hayes, and Debra J. Richardson. 2010. "Design and Evaluation of a Computer Science and Engineering Course for Middle School Girls." In *Proceedings of the 41st*

ACM Technical Symposium on Computer Science Education, 234–38. SIGCSE '10. New York, NY, USA: ACM. doi:10.1145/1734263.1734344.

Margolis, Jane, and Allan Fisher. 2003. *Unlocking the Clubhouse: Women in Computing*. MIT press. http://books.google.com/books?hl=en&lr=&id=StwGQw45YoEC&oi=fnd&pg=PR7&dq=margolis+fisher+unlocking&ots=nnH6KXe5tH&sig=gjxlfMoynNhJF6t-6WDpZe5_xVk.

McDowell, Charlie, Brian Hanks, and Linda Werner. 2003. "Experimenting with Pair Programming in the Classroom." In *ACM SIGCSE Bulletin*, 35:60–64. ACM. http://dl.acm.org/citation.cfm?id=961531.

McDowell, Charlie, Linda Werner, Heather E. Bullock, and Julian Fernald. 2003. "The Impact of Pair Programming on Student Performance, Perception and Persistence." In *Proceedings of the 25th International Conference on Software Engineering*, 602–7. IEEE Computer Society. http://dl.acm.org/citation.cfm?id=776899.

McDowell, Charlie, Linda Werner, Heather Bullock, and Julian Fernald. 2002. "The Effects of Pair-Programming on Performance in an Introductory Programming Course." In *ACM SIGCSE Bulletin*, 34:38–42. ACM. http://dl.acm.org/citation.cfm?id=563353.

McLeod, S. 2007. "Vygotsky | Simply Psychology." http://www.simplypsychology.org/vygotsky.html.

Milam, Jennifer. 2012. "Girls and STEM Education: A Literature Review." https://wiki.cc.gatech.edu/designcomp/images/d/de/CCDC_Final_Writeup.pdf.

Miller, Robert C., Haoqi Zhang, Eric Gilbert, and Elizabeth Gerber. 2014. "Pair Research: Matching People for Collaboration, Learning, and Productivity." In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 1043–48. CSCW '14. New York, NY, USA: ACM. doi:10.1145/2531602.2531703.

NCWIT. 2015. "NCWIT Promising Practice: Pair Programming (Case Study 1): Retaining Women through Collaborative Learning | Computing Portal." Accessed March 10. http://ensemble-beta.cc.vt.edu/node/6323.

Papert, Seymour. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc. http://dl.acm.org/citation.cfm?id=1095592.

Parmaxi, Antigoni, and Panayiotis Zaphiris. 2014. "The Evolvement of Constructionism: An Overview of the Literature." In *Learning and Collaboration Technologies. Designing and Developing Novel Learning Experiences*, 452–61. Springer. http://link.springer.com/chapter/10.1007/978-3-319-07482-5_43.

Parmaxi, Antigoni, Panayiotis Zaphiris, Eleni Michailidou, Salomi Papadima-Sophocleous, and Andri Ioannou. 2013. "Introducing New Perspectives in the Use of Social Technologies in Learning: Social Constructionism." In *Human-Computer Interaction–INTERACT 2013*, 554–70. Springer. http://link.springer.com/chapter/10.1007/978-3-642-40480-1_39.

Resnick, M. 1996. "Distributed Constructionism." In , 280–84. International Society of the Learning Sciences. http://dl.acm.org/citation.cfm?id=1161135.1161173.

Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, et al. 2009. "Scratch: Programming for All." *Communications of the ACM* 52 (11): 60–67.

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. IDC '05. New York, NY, USA: ACM. doi:10.1145/1109540.1109556.

Resnick, M., and E. Rosenbaum. 2013. "Designing for Tinkerability." *Design, Make, Play: Growing the Next Generation of STEM Innovators*, 163–81.

Resnick, M., and Natalie Rusk. 1996. "The Computer Clubhouse: Helping Youth Develop Fluency with New Media." In *Proceedings of the 1996 International Conference on Learning Sciences*, 285–91. ICLS '96. Evanston, Illinois: International Society of the Learning Sciences. http://dl.acm.org/citation.cfm?id=1161135.1161174.

Shaw, Alan. 1995. "Social Constructionism and the Inner City: Designing Environments for Social Development and Urban Renewal." *Unpublished Ph. D. Dissertation. Cambridge, MA: MIT Media Laboratory*. http://llk.media.mit.edu/papers/shaw-PHD.rtf.

Soper, Taylor. 2014. “Analysis: The Exploding Demand for Computer Science Education, and Why America Needs to Keep up.” *GeekWire*. http://www.geekwire.com/2014/analysis-examining-computer-science-education-explosion/.

Tannen, Deborah. 1991. *You Just Don’t Understand: Women and Men in Conversation*. Virago London. http://www.sheltonstate.edu/userfiles/File/faculty/a%20wible/scan0001.pdf.

Teasley, Stephanie, Lisa Covi, Mayuram S. Krishnan, and Judith S. Olson. 2000. “How Does Radical Collocation Help a Team Succeed?” In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, 339–46. ACM. http://dl.acm.org/citation.cfm?id=359005.

Twidale, Michael. 2013. *Over-the-Shoulder Learning: B Supporting Brief Informal Learning Embedded in the Work Context*. Graduate School of Library and Information Science. http://people.lis.illinois.edu/~twidale/pubs/otsl1.html.

Twidale, Michael B., X. Christine Wang, and D. Michelle Hinn. 2005. “CSC: Computer Supported Collaborative Work, Learning, and Play.” In *Proceedings of Th 2005 Conference on Computer Support for Collaborative Learning: Learning 2005: The Next 10 Years!*, 687–96. CSCL ’05. Taipei, Taiwan: International Society of the Learning Sciences. http://dl.acm.org/citation.cfm?id=1149293.1149384.

Utting, Ian, Stephen Cooper, Michael Kölling, John Maloney, and Mitchel Resnick. 2010. “Alice, Greenfoot, and Scratch–a Discussion.” *ACM Transactions on Computing Education (TOCE)* 10 (4): 17.

Weaver, Alfred C. Alf, and Jane Chu Prey. 2013. “Fostering Gender Diversity in Computing.” *Computer* 46 (3): 0022–0023.

Webb, Heidi C., and Mary Beth Rosson. 2011. “Exploring Careers While Learning Alice 3D: A Summer Camp for Middle School Girls.” In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 377–82. ACM. http://dl.acm.org/citation.cfm?id=1953275.

Williams, Laurie A., and Robert R. Kessler. 2000. “All I Really Need to Know about Pair Programming I Learned in Kindergarten.” *Communications of the ACM* 43 (5): 108–14.

Williams, Laurie, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. 2000. "Strengthening the Case for Pair Programming." *IEEE Software* 17 (4): 19–25.

Wills, G. B., H. C. Davis, and E. C. Cooke. 2004. "Paired Programming for Non-Computing Students." http://eprints.soton.ac.uk/259658/.

Zweben, Stuart, and Betsy Bizot. 2013. "2012 Taulbee Survey Strong Increases in Undergraduate CS Enrollment and Degree Production; Record Degree Production at Doctoral Level." *COMPUTING* 25 (5). http://www.cra.org/uploads/documents/resources/crndocs/2012_taulbee_survey.pdf.