Learning Java in a New York City Immigrant Engineer Retraining Program

Suzanna Schmeelk*, Fred Fontaine†, Larisa Ackerman‡ and Alfred Aho§
*Pace University, Seidenberg School of Computer Science and Information Systems,
New York, New York. Email: sschmeelk@pace.edu
†Cooper Union, Albert Nerken School of Engineering
New York, New York. Email: fred@cooper.edu
‡CAMBA, Brooklyn, New York. Email: LarisaA@camba.org
§Columbia University, Computer Science Department
New York, New York. Email: aho@cs.columbia.edu

***Abstract*—This research explores the psychology of programming and the pedagogical environment in a certificate granting urban immigrant engineer retraining program in New York City. The program is aimed at teaching under-represented immigrant engineer students to learn how to program in the Java programming language. The programming concepts and the fostered pedagogical environment were implemented in three-hour evening sessions over 15 weeks in which the students were encouraged to develop programming communities while working on computational thinking concept strands. The research findings that we report are threefold. First, we report on how we fostered building programming concepts into the curriculum into a set of activities specifically designed for an immigrant engineer retraining program with students ranging in backgrounds. We found that at that the program curriculum must be flexible enough for student learning regardless of the fact that a student may miss sessions. Second, we report on how an effective pedagogical environment, which fosters student-centered learning, was promoted so that the students could construct their own meanings of the programming concepts. Third, we report on implementation strategies unique to a retraining program, such as specific environmental constraints as well as how sessions were partitioned into components that fostered computational thinking while learning Java. Our findings provide unique insights into intervention constraints for an urban retraining program which can be used to guide and inform further retraining computer learning program research.**

I. Introduction

The Cooper Union's Retraining Program for Immigrant Engineers is designed to provide engineers the opportunity to update their skills and work in their chosen fields by offering them professional courses and job placement assistance free of charge. The program includes courses in all STEM domains. Specifically, the program offers retraining in computer science, information technology, chemical engineering, mechanical engineering, electrical engineering and civil engineering. All courses are scheduled around the needs of working professionals during weekday evenings and weekends during the day. The engineers must first apply to the program and in many cases already hold degrees.

The Cooper Unions Retraining Program for Immigrant Engineers is managed by Cooper Union and by the nonprofit Workforce Development Department at CAMBA in New York City. The program was started in 1987 by the Scientists Division of Bnai Zion to help scientists and engineers who immigrated to the United States find jobs. Recently, the Robin Hood Foundation has been providing additional support for the engineers. Since the program started in 1987, the Retraining Program for Immigrant Engineers has taught over 4,700 students from six different continents with extremely diverse backgrounds. The program has helped more than 60% of the program students find jobs. The program provides a certificate upon completion of the classes.

When we started teaching Java in the retraining program, we found very little, if any, research on the topic in the field. Our research is thus threefold. First, we develop an intensive and broad curriculum for engineering students to learn to program over a fifteen-week evening program. Our curriculum was designed to engage students of all programming backgrounds and experience levels regardless of missing a few sessions. In addition, the curriculum prepares students for Java Programming Oracle certifications. The broad fifteen week curriculum captures many important topics learned over three years throughout traditional computer science masters degree programs. Second, we report on establishing an effective pedagogical environment which fosters student-centered in-class learning. We report on student learning, computational thinking development and intervention design. The semester long learning manifests into any of the following: a Cooper Union certificate, and Oracle certification and/or an individual program that each student is encouraged to write. The students were encouraged to work at their own pace and openly discuss their ideas. Their individual final program exemplifies the culmination of their semester learning. Third, we report on the unique environmental constraints within an immigrant engineer retraining program. The knowledge for the unique environmental constraints are not discussed in literature at large and can be used to guide further retraining program research.

II. Related Research

Doyle et al. [1] [2] discuss programming experiences for immigrant middle school students in the Northern Kentucky

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

and Greater Cincinnati regions. Their Engagement, Capacity, Continuity (ECC) framework was used to engage students into becoming technology constructors (rather than consumers) in local weekend workshops. ECC was developed to deal with under-representation issues in STEM, according to the authors. In the workshops the students built computers, built robots, and built recipes using computational thinking. The entire family was encouraged to participate.

Schuurink and de Vries [3] relate an immigrant training program in the Dutch society as immigrants in 2009 were required by the government to pass an electronic integration exam pertaining to the Dutch language and culture. The exam is intended for the immigrant to become an active Dutch citizen. The researchers note some difference in training immigrants such as: adult didactics, andragogy, test anxiety, cultural background and accessibility.

Susan O'Donnell [4] reports on a study across five European countries finding that students experiencing disadvantage, which includes immigrant populations, should include IT training/education as one element in a broader pedagogy to help move people into sustainable employment.

Fawcett et al. [5] discuss the use of design thinking to empower United States ethnic minority immigrant youth into a two-phase multi-day workshop called *teen design days*. According to the authors, "These workshops are designed to gain an understanding of how and why immigrant and refugee youth act as Info-Mediaries to members of their social networks, especially their ethnic communities (and elders), and how they can be supported through technology and information services."

Vanbaelen and Harrison [6] argue that lifelong learning "is an efficient way out of poverty and a portal to (re)enter society." The paper examines two lifelong learning projects in Belgium and emphasizes the importance for highly flexible instructors in terms of preparation. The core pillars across the two projects were: language, mathematics, information and communication technologies (ICT), society, job searching and employment adjustments.

Tu and Chen [7] report on using cloud based e-learning systems to teach Chinese language skills to Taiwanese immigrants using games.

West and Bogumil [8] present an opinion article in the Communications of the ACM on the competition for IT jobs in the global labor market during the early 2000s. The authors relate the world wide economic necessity for understanding immigrant issues with respect to IT talent. Finally, another opinion-based article is presented by Ilshatovna and Krieger [9] as they report on immigrants' adaptation through education in Russia and Germany.

III. Intervention Design

In this section, we describe the in depth design choices of our weekly evening retraining interventions as informed by extremely diverse student backgrounds, diverse student time constraints outside the classroom, and diverse student motivations for attending the course. When we started the evening intervention research, there was little, if any, literature informing researchers on either successful intervention design or unique environment constraints. This research paper attempts to fill the gap in literature to address issues unique to the evening urban environment which need to be understood in depth in order to prepare effective teaching interventions.

An effective pedagogical environment has been shown to influence the overall learning outcomes. For example, Ball et al., [10] showed that many mathematics curricula emphasized the learning of algorithms and procedures in mathematics. In such environments, as cited by Erlwanger [11], [12], Skemp [13], Kamii and Dominck, [14] and Mueller, Yankelewitz and Maher [15] students are expected to relinquish their own intuitions and disconnect content from the underlying concepts. The procedural learnings help students achieve in short term goals but in many cases do not help them with understanding underlying mathematics concepts. The traditional teaching disconnect can be a burden to students going forward in later courses and society at large since they really have never been encouraged to reason about the subject topics and may never have even understood many fundamental underlying concepts.

Knowing these traditional weaknesses in educational pedagogy, we wanted to create an environment free of trite processes and procedures. Instead, we wanted to create an environment that encouraged innovation and the learning of concepts while enabling students to continue their studies even when they missed a session. We designed interventions by organizing high level concepts to introduce to students each week in an open environment where students are encouraged to ask questions and discuss ideas with each other.

The concepts were reinforced through a weekly tri-instruction model: syntax lesson, semantic lesson and a software engineering lesson. The syntax lesson explains the structure of Java statements as found in various Java Language Specifications. The semantic lesson explains the meaning and functionality behind the Java language also explained in the Java Language Specifications. The software engineer lessons explain the software life cycle and current industry development trends. Specifically, the aforementioned known pitfalls in education led us to create a pedagogical environment formed around individual student learning, student team building and open environments while taking advantage of interactive learning tools that they could use outside the classroom. The students were encouraged to work in groups and use personal representations to construct their final project. We describe each component of the model in the following subsections and sections.

A. Classroom Leadership

The leadership methods used in the classroom are extremely important for fostering durable learning. Classroom leadership can transpire from multiple perspectives such as teacher instruction, group work, classroom guest visits, online instruction as well as leading sessions in alternative venues (e.g. Startups, Labs, Companies, and Schools). Research has reported, such as Tenenberg et al., [16], that bringing computing

professionals into the student learning experience can improve learning success. The Cooper Union Albert Nerken School of Engineeringand CAMBA sponsor scientific professionals to teach within the Engineering Retraining Program. Classes are either led on site at Cooper Union, CAMBA or New York City sponsored venues. Our pedagogical environment was thus designed to be led by computing and education leaders with diverse backgrounds and experiences in scholastic venues within Manhattan and Brooklyn.

B. Inquiry

Student learning is not a linear process. Our choice of inquiry is supported in part by Simon's research [17] on student learning trajectories. Simon uses the metaphor of traveling to describe learning and states: "The path that you travel is your [actual] trajectory. The path that you anticipate at any point is your *hypothetical trajectory*." Teaching is a planned activity but must be flexible enough to accommodate classroom divergence and individual differences. More recent research refers to learning landscapes to capture the breadth of learning variations [18].

Research has shown that student learning guided by inquiry produces more durable learning. Inquiry is an essential component for developing student reasoning [18]. Fostering an environment where student inquiry leads their learning and student questions are encouraged is essential for students to construct their understanding and build their own learnings. As such, developing open situations (Fosnot and Dolk, [18]), tasks (Mueller, Yankelewitz and Maher, [19], [20]) and interactions with social/experimental environments [21], have been shown to be effective for building long-term understandings. This research guided the instruction choices we made during the sessions. We found that in our session software engineering task, having students personally modify well written programs every week benefited their "big picture" conceptional connections when given a very short period of time to learn within. We found that novice programmers when given a small programming representation within a short interval could more quickly expand the representation than develop a program from scratch in that interval. In fact, requiring new Java programmers to develop a program without guiding examples can lead to student shutdown where if they could not do it, they would turn to do other activities.

C. Classroom Environments

Each session is designed to foster a unique classroom environment where students can openly discuss programming questions while independently working on tasks. Students are encouraged to bring their own laptops to practice in class exercises. Topics are introduced to students and then the students are asked to work on related tasks. Students are given detailed sample programs related to the topics of the day. They then work on related tasks and are encouraged to work with other students in the class. Students are encouraged to further reinforce their learning by working on the new topics between sessions either independently or as teams. We help students put their final project on the Internet (e.g. on GitHub or another repository) where they can reference it on their resumes and/or work on it with other individuals.

Our intervention design decision was based in part on the fact that building a classroom environment and community is an essential component for success in the 21st century team-lead industrial and academic environments. These realities have been reported by Chinn et al. [22], Huss-Lederman et al. ([23] and [24]), and Carter et al. [25]. These researchers reported on the need for collaborative learning and problem-based learning to be adopted into computer science classrooms. They found that an open environment where students could freely collaborate together benefited the student programming skills.

We encouraged students to openly seek advice to their individual questions as well as air their concerns to the group while working on individual tasks and personally meaningful final projects. We foster this environment by issuing a group congress at the start and end of every session and then throughout the learning tri-model, we would encourage students to work with individuals while working on their own personally meaningful programs.

D. Computational Thinking

Computational thinking is a way to construct programs useful in the real world [26]. We designed our sessions to foster computational thinking by giving students mini-real world problems which they would need to be mapped down into a computer programming language representation. Our instructional choices were selected to develop durable programming skills. Research by Helminen et al. [27] and a panel discussion by Bryant et al., [28] show examples of this fact. In Jeannette Wing's *computational thinking* opinion paper [26], she states, "Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking. [26]" One of the hardest components of learning computational thinking is the effective exposure of the topic. Our interventions were designed around student personal interests. We will describe the representations we chose for the computational thinking in the following section.

1) Representations: Our session language was designed around student interests. The students discussed their backgrounds during the first week of class to gain an understanding about their needs, requirements and interests in programming. As we progressed over the term, we followed the styles for clinically interviewing students [29], [30] to gain some understanding of how students were progressing.

In our weekly interventions we would openly discuss student personal meaningful representations which would be the focus of their weekly software engineering activity or software engineering final project. Our methodology reflects

the research of Francisco and Maher [31], [32], Schmeelk [33] and Suthers and Hundhausen [34]. One of the things the students expressed much satisfaction with was the diversity among the class in weekly discussion topics and activities.

Considering representations, we designed a task for the final project to be a useful Java application which would manage their business and personal contacts. The software engineering task was for the student to focus on ideas that personally resonated with the student engineer, their friends, their families, their business associates and their acquaintances. They could optionally write a program for whatever set of representations that resonated with them personally. This methodology reflects the work of Liebenberg et al., [35], work on relevance. Relevance of academic coding exercises, as used by Liebenberg et al., examines coding in academics with how applicable they are to the needs of and interests of students and society and finds higher student involvement and interest when more applicable tasks are chosen.

2) Understanding: Our session design was informed by the work of Davis and Maher [36] to help students build durable programming concepts [37]. Davis describes *Understanding "Understanding"*, [38], in his research when he states, "Put in its starkest terms, this theory postulates that one gets the feeling of *understanding* when a new idea can be fitted into a larger framework of previously assembled ideas. A metaphor that reflects this quite well is the notion that one assembles ideas in one's own mind much as one assembles a jigsaw puzzle." Davis argues against superficial verbal learning and following algorithmic recipes. Our weekly sessions were informed by the curriculum but were used for conceptual purposes rather than to force students into following a superficial procedural learning recipe. This was specifically chosen by using assimilation paradigms as published by Davis and Maher [39]. In an *assimilation paradigm* new learnings are built on powerful existing representations through the use of mathematically isomorphic tasks which we used in our weekly interventions by using personally meaningful activities for which the students could relate and understand. Thus, we adapted the notion of an assimilation paradigm by developing programming tasks based on students prior representations (examples: games they play, things they do, people they know, asking the students for metaphors, etc.) when we introduced students to new programming ideas.

3) Java Selection: There are three main reasons why Java was selected for the language to learn in our weekly interventions. First, Java is clearly used extensively in current research methods in many programming fields. Second, Java is one of the main languages behind many web service applications. In many cases, the applications transfer encrypted data to end users world wide. Third, it is the fundamental underlying language of the Android mobile application development. Java remains a heavily used development language worldwide.

IV. Session Structures

Sessions took place in weekly three-hour intervals for approximately thirty students of extremely diverse ethnicities and genders in an evening program. Each session was divided into at least four high-level components over fifteen units. First, we would discuss the prior weeks learning, if any. This open discussion encouraged students to recall learnings from the prior week which they may have extended outside the classroom as well as having students listen to other student questions and ideas. Literature has refered to this methodology as a *congress* [18]. Second, we would discuss the topics for the week by asking the students about things they personally do. We would introduce technical language like *control flow*, *function*, *data structures*, *algorithm* and *software engineering* through terms that they use in their daily lives. We would also interlace noteworthy topics from the industry news into the discussion. Third, we would work on learning new tools (IDE and command-line) and connecting the concepts through sample example programs. The sample programs were structured at different difficulty levels for students at different points in their learnings. The students could work through understanding them at their own pace in class on their laptops as well as outside of class. We structured the programs to accommodate student pace variations. Finally, students were encouraged to start their final project during the first unit. As they learned new ideas, they were then encouraged to refactor their code to incorporate the new ideas into their final project.

V. Retraining Curriculum

We structured our evening curriculum around basic computing notions. The concepts covered in the class included compiler design, operating system concepts, networking, secure coding, databases and data structures. We wanted students to understand how programming languages relate to real-world problem solving, as well as, understanding fundamental language concepts. We started by showing how high-level programming concepts fit into computing. We then spent two sessions introducing fundamental Java syntax and control flow. The next session we spent on software engineering as the students started to design their own program. The next few sessions were focused on fundamental programming language ideas that students might use as they built their own program. These topics included inheritance, encapsulation, abstract methods, interfaces, generics, exception handling and data structures. The last few sessions revolved around advanced concepts such as multithreading, databases, JUnit testing and networking. The final session included presentations for any students that wanted to share their final project.

VI. Oracle Certification

The course is designed to prepare novice and intermediate Java programmers to become advanced Java programmers over 15 weeks. The course provides students with a certificate upon successful completion of the course. The course curriculum pulls from necessary topics included in both a Computer Science Bachelors of Science and a Computer Science Masters of Science as well as pulling from topics required for the students to pass additional and optional Oracle Java Programming certification courses.

As training to prepare for the Oracle Java exams can cost thousands of dollars, the costs are too high for many immigrant students. In order to supplement the Cooper Union Certificate, the course is aimed as providing students the training to help them prepare to pass the "Java SE 7 Programmer I" Associate Level Certification exam given by Oracle. The course also prepares students for everything on the "Java SE 8 Programmer II" Professional Level Certification exam except Lambda expressions. Specifically, we include the following topics into the course: Java Basics, Working With Java Data Types, Using Operators and Decision Constructs, Creating and Using Arrays, Using Loop Constructs, Working with Methods and Encapsulation, Working with Inheritance, Handling Exceptions, Java Class Design, Advanced Java Class Design, Generics and Collections, Exceptions and Assertions, Use Java SE 8 Date/Time API, Java I/O Fundamentals, Java File I/O, Java Concurrency, Building Database Applications with JDBC, Localization. We are working into integrating Lambda expressions into the course to introduce Java's capabilities for functional programming. Finally, we are working to introduce Java 9 topics into the course (e.g. modules, HTTP2, jshell, etc.) and will continue to update the course as newer version of Java are released.

VII. Evaluating Understandings

Understanding was evaluated based on classroom discussions and feedback from independently developed and written programs. The students all were encouraged to independently develop a contact application written in Java on any topic of their choosing. The application could include control flow, input, output, data structures, methods, classes, abstract methods, interfaces, database connections, networking, threading, exceptions, data structures, testing, API creation and documentation. At the beginning of every session, students were asked to recall what they had learned both in class during the prior session and out of class while reviewing the material. Students regularly raised their hands to ask questions and/or to add comments to what was learned in prior session which exemplifies the Pirie and Kieren model [40] of learning by folding back and thickening understandings.

VIII. Lessons Learned from A Retraining Environment

We, as educators and researchers, have learned a great deal from our immigrant engineering retraining experience. First, we learned that many students had had some programming exposure to languages such as COBOL, Fortran, SQL and were engineers with deep analytical skills. There are usually a few students in each section whom have programmed in Java before. Some concepts in the Java language were new to many students such as the virtual machine, application programming interface and naming conventions.

Second, the extremely diverse student backgrounds both in education as well as country of origin creates a beautifully idea-rich and culturally-rich classroom. Since all the students are engineers, the level of knowledge they bring to the classroom is extremely advanced. In addition, the motivation for taking the class is extremely diverse. We did learn that some of the students are working, many are learning English and some are not yet completely familiar with the New York culture. These constraints require a flexible and innovative program where these traditional barriers do not cause student attrition. In fact, many student may sit in on the class even if they know the Java concepts simply to become fluent on the topics in English. The motivations for attending class are extremely diverse as well. In the class, we have had lawyers, Ph.D.s and parents (wanting to teach their children) who simply want to learn about programming and have no intention to pass the Oracle certification. In the same class, we have B.S. and/or M.S. Computer Science students who already know a great deal of the content and plan to get certified. The class pedagogy has to be adjusted to suit a wide range of attendance motivations so that everyone benefits in some way from the course.

Third, we found that we need to expose the students to IDEs (e.g. NetBeans, Eclipse and IntelliJ) as well as command-line utilities for them to understand how a programming language connects with an Operating System. Our students brought their own programming devices. The BYOD environment helped students set-up their systems for when they would study outside class. The BYOD paradigm offers students some level of consistency between the classroom and their out of class study. Any student who could not bring their own device for various reasons (e.g. work schedule, commute, etc.) was encouraged to work with a partner who had in fact brought a laptop.

Fourth, we developed take-home packets for students with software examples handed out weekly on a USB. The examples assist students when they are at home, if they continue their programming exercises. This is intended to help students who do not have anyone locally that can help them learn to program. The classroom can help the students foster a local computing community.

Fifth, some of our students recommended online code tools. Much of the online training is not complete and in many cases is not specific to immigrant engineers who are learning English. The students do request to have online learning tools (e.g. CodeAcademy, Coursera, Lynda, KhanAcademy, etc.) to reference. However, many students expressed that they liked the in person meetings for social interactions and to lean from their peers.

Sixth, we would encourage a repository of student learning in programming to be constructed as the Video Mosaic Collaborative (VMC) has proven to be quite successful in studying Mathematics Education learnings, as shown by [41], [42], [43].

Finally, our evening curriculum can be viewed online, replicated and expanded.

IX. Conclusions

The weekly evening New York City immigrant engineer retraining environment creates significantly unique programming

parameters, conditions and restrictions which are not present in other learning environments. Until this research, we were unable to identify many, if any, papers discussing the uniqueness of an urban immigrant engineer retraining program. Learning to program is more than simply memorizing a process or following a procedure which must be obeyed. It is essential for students to ask questions and apply their learning to real-world programs with which they personally resonate. As such, durable learning must be more than superficial memorization of processes and procedures [18]. They must be able to take their learning and walk into new unfamiliar situation and be able to fall back on conceptual ideas of what a programming language can do so that they will succeed even when they encounter new programming territories.

Our curriculum was designed to provide a framework for students in which they can independently software engineer their own complex programs in Java. As shown by Malmi et al., [44], when they analyzed the Computing Education Research literature to "discover what theories, conceptual models and frameworks recent CER buil[t] on." It was shown that there is a broad and vast background on various models and frameworks spanning multiple research generas (psychology, education, etc.) in the literature. This shows the complexity of the field and emphasizes the growth potential.

REFERENCES

[1] M. Doyle, K. G. Kirby, and G. Newell, "Engaging constructions: Family-based computing experiences for immigrant middle school students," *SIGCSE Bull.*, vol. 40, no. 1, pp. 58–62, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1352322.1352158

[2] ——, "Engaging constructions: Family-based computing experiences for immigrant middle school students," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '08. New York, NY, USA: ACM, 2008, pp. 58–62. [Online]. Available: http://doi.acm.org/10.1145/1352135.1352158

[3] E. Schuurink and M. de Vries, "Combining advanced learning technologies in an immigrant educational program," in *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, ser. MindTrek '09. New York, NY, USA: ACM, 2009, pp. 114–117. [Online]. Available: http://doi.acm.org/10.1145/1621841.1621862

[4] S. O'Donnell, "It education and training for disadvantage students: lessons from europe," *IEEE Technology and Society Magazine*, vol. 24, no. 3, pp. 23–31, Fall 2005.

[5] P. Fawcett, K. E. Fisher, A. P. Bishop, and L. Magassa, "Using design thinking to empower ethnic minority immigrant youth in their roles as technology and information mediaries," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '13. New York, NY, USA: ACM, 2013, pp. 361–366. [Online]. Available: http://doi.acm.org/10.1145/2468356.2468420

[6] J. Harrison and R. Vanbaelen, "Lifelong learning as a steppingstone to entrepreneurship and innovation," in *2016 IEEE International Professional Communication Conference (IPCC)*, Oct 2016, pp. 1–4.

[7] C. C. Tu and A. P. Chen, "Building a learning games network in cloud learning platform based on immigrant education," in *2011 International Conference on Advances in Social Networks Analysis and Mining*, July 2011, pp. 746–750.

[8] L. A. West and W. A. Bogumil, "Immigration and the global it work force," *Commun. ACM*, vol. 44, no. 7, pp. 34–38, Jul. 2001. [Online]. Available: http://doi.acm.org/10.1145/379300.379307

[9] R. I. Zinurova and W. Krieger, "Educational technologies of immigrants' adaptation in russia and germany," in *2013 International Conference on Interactive Collaborative Learning (ICL)*, Sept 2013, pp. 536–538.

[10] D. L. Ball, C. Hoyles, H. N. Jahnke, and N. Movshovitz-Hadar, "The teaching of proof." in *Proceedings of the international congress of mathematicians*. Higher Education Press, 2002, pp. 907–920.

[11] S. H. Erlwanger, "Bennys conception of rules and answers in ipi mathematics." *The Journal of Childrens Mathematical Behavior.*, vol. 1(2), pp. 7–26, 1973.

[12] K. R. Leatham and T. Winiecke, "The case of the case of benny: Elucidating the influence of a landmark study in mathematics education," *The Journal of Mathematical Behavior*, vol. 35, pp. 101 – 109, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0732312314000315

[13] R. R. Skemp, "Relational understanding and instrumental understanding." *Mathematics teaching.*, vol. 77, pp. 20–26, 1976.

[14] C. Kamii and A. Dominck, *The harmful effects of algorithms in grades 1 through 4.* Reston, VA: The National Council of Teachers of Mathematics., 1998.

[15] M. Mueller, D. Yankelewitz, and C. Maher, "Rules without reason: Overcoming students obstacles in learning." *The Montana Mathematics Enthusiast.*, vol. 17(2/3), pp. 307–320, 2010.

[16] J. Tenenberg, "Industry fellows: Bringing professional practice into the classroom," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, pp. 72–76. [Online]. Available: http://doi.acm.org/10.1145/1734263.1734290

[17] M. Simon, "Reconstructing mathematics pedagogy from a constructivist perspective." *Journal for Research in Mathematics Education.*, vol. 26, pp. 114–145, 1995.

[18] C. Fosnot and M. Dolk, *Young Mathematicians at Work.* Heinemann Publishers. Portsmouth, New Hampshire, USA, 2001.

[19] M. Mueller, D. Yankelewitz, and C. A. Maher, "Promoting student reasoning through careful task design: A comparison of three studies." *International Journal for Studies in Mathematics Education.*, vol. 3, p. , 2010.

[20] D. Yankelewitz, M. Mueller, and C. A. Maher, "A task that elicits reasoning: A dual analysis," *The Journal of Mathematical Behavior*, vol. 29, no. 2, pp. 76 – 85, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0732312310000064

[21] A. H. Schoenfeld, "Beyond the purely cognitive: Belief systems, social cognitions, and metacognitions as driving forces in intellectual performance." *Cognitive science.*, vol. 7(4), pp. 329–363, 1983.

[22] D. Chinn and K. Martin, "Collaborative, problem-based learning in computer science," *J. Comput. Sci. Coll.*, vol. 21, no. 1, pp. 239–245, Oct. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1088791.1088833

[23] S. Huss-Lederman, D. Chinn, and J. Skrentny, "Serious fun: Peer-led team learning in cs," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '08. New York, NY, USA: ACM, 2008, pp. 330–331. [Online]. Available: http://doi.acm.org/10.1145/1352135.1352249

[24] C. Hundhausen, A. Agrawal, D. Fairbrother, and M. Trevisan, "Does studio-based instruction work in cs 1?: An empirical comparison with a traditional approach," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, pp. 500–504. [Online]. Available: http://doi.acm.org/10.1145/1734263.1734432

[25] A. S. Carter and C. D. Hundhausen, "The design of a programming environment to support greater social awareness and participation in early computing courses," *J. Comput. Sci. Coll.*, vol. 31, no. 1, pp. 143–153, Oct. 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831373.2831399

[26] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006. [Online]. Available: http://doi.acm.org/10.1145/1118178.1118215

[27] J. Helminen, P. Ihantola, V. Karavirta, and L. Malmi, "How do students solve parsons programming problems?: An analysis of interaction traces," in *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ser. ICER '12. New York, NY, USA: ACM, 2012, pp. 119–126. [Online]. Available: http://doi.acm.org/10.1145/2361276.2361300

[28] R. Bryant, D. Chinn, G. Hauser, M. Folsom, and S. Wallace, "Computational thinking: What is it, how is it relevant, who's doing what with it?" *J. Comput. Sci. Coll.*, vol. 25, no. 1, pp. 170–171, Oct. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1619221.1619255

[29] C. A. Maher and R. Sigley, *Task-Based Interviews in Mathematics Education*. Dordrecht: Springer Netherlands, 2014, pp. 579–582.

[30] H. Ginsburg, "The clinical interview in psychological research on mathematical thinking," vol. 1(3), pp. 4–11, 1981. [Online]. Available: http://www.jstor.org/stable/40247721

[31] J. M. Francisco and C. A. Maher, "Conditions for promoting reasoning in problem solving: Insights from a longitudinal study." *The Journal of Mathematical Behavior.*, vol. 24(3), pp. 361–372, 2005.

[32] ——, "Teachers attending to students mathematical reasoning: Lessons from an after-school research program." *Journal of Mathematics Teacher Education.*, vol. 14 (1), pp. 49–66, 2011.

[33] S. Schmeelk, *An Investigation of Fourth Grade Students Growing Understanding of Rational Numbers. Unpublished Doctorial Dissertation.* New Brunswick, NJ: Rutgers University, Jan 2010.

[34] D. D. Suthers and C. D. Hundhausen, "The effects of representation on students' elaborations in collaborative inquiry," in *Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community*, ser. CSCL '02. International Society of the Learning Sciences, 2002, pp. 472–480. [Online]. Available: http://dl.acm.org/citation.cfm?id=1658616.1658682

[35] J. Liebenberg, M. Huisman, and E. Mentz, "The relevance of software development education for students," *IEEE Transactions on Education*, vol. 58, no. 4, pp. 242–248, Nov 2015.

[36] R. B. Davis and C. A. Maher, "Chapter 5: What do when we do mathematics?" pp. 65–210, 1990.

[37] C. A.Maher and K. Weber, "Representation systems and constructing conceptual understanding." *Special Issue of the Mediterranean Journal for Research in Mathematics Education.*, vol. 9(1), pp. 91–106, 2010.

[38] R. B. Davis, "Understanding 'understanding.'," *Journal of Mathematical Behavior.*, vol. 3, pp. 225–241, 1992.

[39] R. B. Davis and C. A. Maher, *How Students Think: The Role of Representations.* Hillsdale, NJ: Lawrence E. Earlbaum Associates., 2003.

[40] S. Pirie and T. Kieren, "Growth in mathematical understanding: How can we characterize it and how can we represent it?" *Educational Studies in Mathematics.*, vol. 26, pp. 165–190, 1994.

[41] S. Schmeelk and R. Sigley, "New tools for research: Using the video mosaic collaborative." in *Proceedings of the American Society for Engineering Educaiton (ASEE).* ASEE, 2012, pp. 25.975.1–25.275.5.

[42] G. Agnew, C. M. Mills, and C. A. Maher, "Vmcanalytic: Developing a collaborative video analysis tool for education faculty and practicing educators," in *2010 43rd Hawaii International Conference on System Sciences*, Jan 2010, pp. 1–10.

[43] C. E. Hmelo-Silver, C. A. Maher, A. Alston, M. F. Palius, G. Agnew, R. Sigley, and C. Mills, "Building multimedia artifacts using a cyber-enabled video repository: The vmcanalytic," in *2013 46th Hawaii International Conference on System Sciences*, Jan 2013, pp. 3078–3087.

[44] L. Malmi, J. Sheard, Simon, R. Bednarik, J. Helminen, P. Kinnunen, A. Korhonen, N. Myller, J. Sorva, and A. Taherkhani, "Theoretical underpinnings of computing education research: What is the evidence?" in *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ser. ICER '14. New York, NY, USA: ACM, 2014, pp. 27–34. [Online]. Available: http://doi.acm.org/10.1145/2632320.2632358