Scratch Analyzer

Transforming Scratch Projects into Inputs Fit for Educational Data Mining and Learning Analytics

Jeff Bender
Computer Science | Software Systems
Columbia University
New York, New York, U.S.
jrb2211@columbia.edu

***Abstract* — At an unrivaled and enduring pace, computing has transformed the world. The resulting global economic circuitry demands from its operators more than reading, writing, and arithmetic: a fourth foundation, computational thinking. This new baseline need arises in an era of incommensurate education reform focused more on assessment than durable learning. Meanwhile, computing's proliferation has empowered students with versatile programming, enabling passion-driven discovery that engages the minds that schools too frequently fail to attract or satiate. Educators increasingly acknowledge that game-based learning activates student motivation and engagement, but introducing games into classrooms often proves too taxing for overburdened teachers. Scratch Analyzer displaces some of this strain by enabling the automation of time-consuming learning and instructional responsibilities through the use of emerging techniques in educational data mining and learning analytics. As a big data toolkit for learning-system designers who aim to infuse computational thinking within K-12 curricula, Scratch Analyzer positions Scratch, the eminent visual programming environment for novices, as a game-based learning platform interconnected with an attainable future in which students engage in properly-paced, enjoyable, adaptive, and personalized mastery of the computational thinking concepts driving economic and social progress.**

Educational data mining, learning analytics, computational thinking, learning sciences, game-based learning, K-12 curricula.

I. Introduction

"For the most part, instructional designers know little about game development and video game developers may know little about training, education and instructional design" [1].

Two years after the first pixelated video game, Spacewar!, blipped onscreen at MIT in 1961 [2], determined researchers began exploring the potential of video games for learning [3], but a persisting chasm between instructional design and game development continues to roil attempts at emancipating classrooms from their industrial era origins [4]. U.S. K-12 education has stagnated, increasingly requiring students to memorize stale facts from dusty paper artifacts in preparation for regurgitation during standardized tests [5]. Simultaneously, video games have exploded from their roots in geek culture into a predominate mainstream media [6], eclipsing the annual revenue of the U.S. movie and music industries in 2005 and 2007, respectively [7, 8, 9]. And at a parallel, exponential clip, society has demanded from its citizens technological proficiency that no longer simplifies to assigning children the task of programming blinking VCRs [10]. This report first introduces challenges in U.S. K-12 education, opportunities in game-based learning (GBL), and the pressing salience of computational thinking (CT), in order to expose under-recognized yet ripe opportunities for big data analytics to contribute to innovative classroom reconfiguration that prioritizes engaged learning imbued with CT. Harmonizing these disciplines is critical because, as K-12 education enters an era of transformation during which dynamic devices surely will replace static textbooks, the question of whether instruction and curricula will leverage the affordances of these advances to prepare students adequately for our transition from a Knowledge Society to a Creative Society remains unsettled [11, 12]. More simply, will today's learning equip students to fabricate innovative solutions to the challenges of tomorrow?

Although without providing a definitive response, the core of this report examines an open-source toolkit aimed at aiding learning-system designers in achieving this lofty goal. The modular middleware suite, Scratch Analyzer, distills Scratch projects into inputs fit for educational data mining (EDM) and learning analytics (LA), most broadly defined as big data analytics applied in the educational arena, with EDM generally involving more automation, and LA typically focusing more closely on informing teachers and learners [13]. Since thousands of "Scratchers" using the novice programming environment, Scratch, create games, animations, stories, and more, and share their constructions with peers and mentors across a global community on a daily basis [14], interconnecting their constructions to big data infrastructures could uncover unforeseen opportunities for insightful learning. For example, Scratchers who have used similar blocks in recent projects could discover one-another, and their related creations, if a user-based recommendation engine could propose appropriate contacts. Such a system could help Scratchers' understand particular CT concepts from a variety of perspectives, thereby reinforcing initial perceptions and intrinsically motivating deeper collaborative study. Alternatively, big data systems, via clustering and classification, could automate labor-intensive evaluative tasks, freeing teachers' time for more productive problem-solving with individual students. And on a more granular scale, an item-based recommendation engine could enable a

GBL, CT curricula deployed in Scratch to offer adaptive scaffolding by presenting hints about which blocks to try as students encounter difficulty [15]. By connecting the expanding Scratch community to the abundant possibilities of big data analytics, Scratch Analyzer positions Scratch as a powerful platform upon which to implement education innovation.

The remainder of the report is structured as follows. Section 2 provides the aforementioned primers on the challenges in U.S. K-12 education, the affordances of GBL, and the conspicuous pertinence of CT. Section 3 describes Scratch Analyzer's implementation and reviews initial experimental results. Section 4 introduces aspirational future work within the context of the doctoral research agenda, Social Addictive Gameful Engineering (SAGE). Section 5 summarizes to conclude the report.

II. Related Work

A. K-12 Education

> "Schools haven't changed; the world has. And so our schools are not failing. Rather, they are obsolete - even ones that score the best on standardized tests" [16].

Replacing the master-apprentice relationships that sustained the agrarian society of the 18th century, the U.S. K-12 education system we know today developed during the standardization zeitgeist of early industrialization. Education leaders such as Horace Mann envisioned a universal school system that would extend beyond the Jeffersonian ideal of enlightening citizens in order to cultivate a democracy capable of sustainable self-government [17]. In a thriving economy driven by assembly lines, the factory model for schools soon set state-of-the-art norms; a small number of teachers instructed a much larger number of students with sufficient proficiency to produce the reliable labor force that propelled the nation to super-power ascendency [18]. Particularly in secondary school, this educational engine derived structure from the Carnegie unit, which measures academic attainment by time, specifying that students spend 120 hours of contact or class time with an instructor per year, or approximately 45 minutes per day, five days a week, for the 36 weeks of the academic calendar [19]. The somewhat arbitrary elevation of time as assessor permeated to the elementary and primary schools, ultimately inculcating a system of schooling measured preponderantly by instruction hours rather than more meaningful material mastery [20].

Over a century later, education's entanglement with professional and social structure largely endows inertial stasis [5], but the federal reform movement begun in the 1990's has initialized a transition which diminishes time and elevates accountability. The No Child Left Behind Act passed by Congress in 2001, the Common Core State Standards Initiative developed by the National Governors Association and Council of Chief State School Officers in 2009, and the Obama Administration's Race to the Top incentive program of 2010, all advocate that states establish and meet high and increasingly consistent academic standards. Two fiercely debated questions have emerged, however: are standardized tests holistic student assessments, and do they measure durable learning [21]? Rather than collaboratively seek answers, more frequently, apparently competing stakeholders such as teachers unions, local school boards, district administrators, and parents vie for power, control, and influence, while regularly disregarding student voices proven to have positive effects on school success [22, 23]. Once discarded, these youth perspectives degenerate through disinterest, disengagement, and dissatisfaction [24], creating a dystopia which starkly contrasts with the utopian affordances these digital natives discover on devices enabling exploration of personal interests and passions in immersive multimedia Metaverses unmatched by reality [25, 26]. We glimpse at formative research highlighting the efficacy of video games integrated with formal and informal learning environments next.

B. Game-Based Learning

> "Playing the game made you pay attention and read everything" [27].

A classroom of children paying attention and reading everything represents an ideal which K-12 teachers would cheerfully embrace. By describing practically her motivation, the high school freshman quoted above distills the allure of GBL. The seminal scholar in this nascent discipline, James Paul Gee, expands this description without introducing jumbling jargon:

> "Game manuals, just like science textbooks, make little sense if one tries to read them before having played the game. All one gets is lots of words that are confusing, have only quite general or vague meanings, and are quickly forgotten. After playing the game, the manual is lucid and clear because every word in it now has a meaning related to an action-image, can be situated in different context of use for dialogue or action" [28].

Learning scientists might classify such a seemingly straightforward observation as situated embodied learning theory, but the key in this context is the nuclear force with which games motivate students to engage in durable learning. Whether framed as GBL, serious games, gamification, or edutainment, the experience is consistently gameful: goal-oriented, curiosity-driven, failure-fearless, optimistic, and fun [26]. The transposition of game mechanics, the rule-based constructs of interactivity, into curricular frameworks, provides the enjoyably pervasive interchange that keeps attention from wandering [29], inviting students to construct their own explanations actively instead of listening to lectures passively [30]. Adaptive, pedagogically aligned reward structures addict children to learning by streaming instantly gratifying feedback which encourages practice toward mastery [31].

Opponents of this educational innovation argue, however, that many games present an additional mass of complexity which unacceptably forces students to focus on often intricate game mechanics at the expense of standard domain content [32]. This justifiable concern exposes a breach between game and school cultures; traditional gaming practices, such as long play sessions, do not easily fit into Carnegie unit slices. Blurring the boundaries between game mechanics and domain learning might prove the most effective response in formal settings [27], but introducing games at school does not preclude their introduction in informal learning environments. Curricula-aligned after-school programs, technology centers, clubs, camps, libraries, and countless other venues offer interest-driven opportunities for children to explore their passions unleashed within good games [33].

For stakeholders to appreciate fully how these good games could invigorate students and revitalize schools, however, they must recognize the video game as an aesthetic medium plump with its own unique capabilities [34]. The educational goal does not devolve solely into leveraging games to motivate paying attention and reading everything, but rather to suffuse learning through vivid interactivity within immersive virtual worlds [35]. The widely espoused flow theory advanced by positive psychologist Mihály Csíkszentmihályi best characterizes this immersion: intense concentration; clarity of action; immediate feedback; balance between challenge and skills; feeling of control; exclusion of irrelevance from consciousness; distortion of time; and intrinsically rewarding behavior [36]. When augmented by social interaction and learning in games, the even richer conceptualization of social gameflow emphasizes a critical dependency beyond game design: the cooperative context in which students play [37]. Designing effective learning environments thus requires careful consideration of both the curricular artifacts and the emergent community structure [38].

The prospect of socially situated, addictively engaging, gamefully fun learning presents boundless opportunities for K-12 education. If united with the catalyzing force of the United States National Academies (USNA), who in 2006 raised awareness that the decline of science, technology, engineering, and math (STEM) preparation could reduce U.S. economic competitiveness, GBL could reinvigorate engineering in schools by motivating blissfully productive students to overcome pleasurably frustrating challenges [26]. This report focuses on the affordances offered by big data analytics to GBL in one important sliver of the applied possibility space carved from STEM's underserved "E": CT. In the following subsection we introduce this subject cultivated by computer scientists but envisioned for all.

C. Computational Thinking

> "Computational thinking will influence everyone in every field of endeavor. This vision poses a new educational challenge for our society, especially for our children" [39].

If 2006 stoked STEM, it enkindled CT. Writing as the head of the Computer Science Department at Carnegie Mellon University, Jeanette Wing articulated a call for educational action that releases her subject from the constraints of its scholarly seclusion. Stipulating that ubiquitous computing is to today as CT is to tomorrow, she frames CT as a "universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" [40]. Although a consensus definition for the term has remained elusive, CT aims to describe how humans, not computers, think logically, algorithmically, procedurally, analytically, recursively, and creatively, with an engineering-attunement [41]. This hybrid thinking helps us devise models and representations which enable problem solving, data transformation, and system automation [42]. The primacy of conceptualization, not programming, of ideas, not artifacts, frees computer science fundamentals from their constituent levees, allowing strong mental models to flood diverse disciplines both in the sciences and humanities, as well as in in their applied counterparts, medicine, law, and journalism [43]. Of course, computers retain their central role: "Abstractions are the 'mental' tools of computing. The power of our 'mental' tools is amplified by the power of our 'metal' tools. Computing is the automation of our abstractions" [39]. The emphasis on semantics rather than syntax, however, engenders a common language in which to explore the possibilities of computing, and a means by which to confront the dwindling wonderment experienced by digital natives commonly capable of using enveloping technologies, but devoid of digital fluency.

Educational technology evangelist Marc Prensky coined this classification, digital native, in 2001, as means by which to emphasize the manner in which Internet era students process information fundamentally differently from their predecessors [44]. Although accurately differentiating digital natives from their digital immigrant parents by identifying their desire for instant gratification and penchant for parallel processing, multi-tasking, randomly accessing information, hypertext thinking, and networking, he effectively limits his analysis to the extent to which this technologically savvy generation efficiently uses computers, video games, and the Internet [25], a set of competencies otherwise known as computer literacy [45]. Mitchell Resnick, leader of the Lifelong Kindergarten group at the MIT Media Lab, refines Prensky's discourse by focusing on digital fluency, which he argues "requires not just the ability to chat, browse, and interact but also the ability to design, create, and invent with new media" [46]. This spotlight on digital construction naturally elicits an emphasis on learning programming, a skill which substantially expands the range with which students can express themselves [47]. Digital construction thus becomes the setting for students to program while developing problem-solving and design strategies, such as modularization and iterative design, which interlock with non-programming domains.

The resulting interdisciplinary educational experience ultimately underwrites an expansive, yet inclusive, orientation around CT, extending beyond both computer literacy and digital fluency to encompass analytical skills which enhance how we approach problems and how we leverage computing to augment our abilities [48]. Such a sweeping mandate reinforces Wing's compelling vision for how computer science educators, researchers, and practitioners can act to change society's sometimes narrow image of the field; instead of equating computer science with computer programming, thereby isolating widely applicable conceptual competencies within the nooks of spare computer classes, we can expose the urgent need to foster a K-12 computational culture [49]. Existing curriculum standards, infrastructure deficiencies, and limited professional development opportunities for teachers to learn CT all present significant challenges [50], but implementing CT during school hours does not entail slicing a new Carnegie unit out of an already-eaten pie. Programming, for instance, is reflexive with other domains; learning to code while exploring concepts from other subjects can ease the understanding of all involved topics concurrently [51, 52]. We can, therefore, among numerous other examples, demonstrate binary search when teaching how to find the roots of an equation, or introduce optimization when training students in Excel [49]. By consistently integrating CT where it fits in practice in established classroom curricula, we infuse a set of shared attitudes, values, goals, and practices which ultimately catalyze this computational culture.

Fortunately, in the last several years, many organizations have recognized the imperative of introducing CT to K-12 students, and progressive change has begun to impact schooling. The National Science Foundation (NSF) has established standards which formalize CT's core: problem formulation; logically organizing and analyzing data; representing data through abstractions such as models, automating solutions through algorithmic thinking; implementing effective solutions optimally; and transferring the solution to solve a wide variety of problems [53]. The NSF additionally bolsters its commitment through the CS10K Project, which aims to recruit ten thousand teachers prepared to teach computer science in ten thousand schools by 2016. Considering that in 2012, only 2,000 teachers qualified to teach the Advanced Placement CS course, which less than 10% of U.S. high schools offer, the NSF has set aggressive goals [54]. Importantly, however, increasing the quantity of computer scientists remains only a tangential agenda of the broader CT movement [55]. The National Research Council has declared CT a cognitive skill which the "average person is expected to possess", and noted that "students can learn thinking strategies such as computational thinking as they study a discipline" already entrenched [41]. From this perspective, CT does not demand from school stakeholders protracted debate about wrangling time-slices for computer classes; instead, CT infuses within existing curricula [56]. In particular, STEM's momentum provides an apt opportunity: "There is inherently a C (Computing) in STEM. Learning STEM without learning computing is fundamentally inadequate. Learning computing while solving STEM problems, on the other hand, would inevitably foster one's computational thinking ability" [49]. Devising how to seize this opportunity adeptly emerges as a critical arena for pragmatic field research which levers EDM and LA while integrating CT in the formal and informal learning environments inhabited by K-12 students.

III. Implementation & Results

> "Models... have even been able to predict... students' final course grades. This work has been embedded into automated agents that scaffold more effective collaboration and into tools to support instructors in scaffolding their student's collaboration" [13].

A. *System Architecture*

The integration of big data analytics into innovative learning systems presents one viable pathway forward. Since long-term aspirations envision a sizeable suite of software tools facilitating a variety of EDM and LA techniques, Scratch Analyzer employs a modular architecture which affords maximal flexibility. Rather than introducing entangling interdependencies, this design explicitly specifies intermediate input and output formats or interfaces, thereby exposing stable surfaces upon which future components can build. This modularity distinguishes Scratch Analyzer as a set of reusable and reconfigurable components that assist in answering questions as they emerge. As noted in [57], a key strategic benefit of big data analyses involves discovering actionable information without first fully formalizing cogent queries; big data systems, then, must offer the plasticity to meet an assortment of known and unknown needs.

As depicted in Figure 1, Scratch Analyzer provides this plasticity via Java implementations of three components: Scratch Extractor, Scratch Dispatcher, and Scratch Traverser. The choice in programming language and run-time environment aligns with the big data ecosystem consisting of commonly used frameworks such as Hadoop and Mahout [58, 59]. For example, the Apache Lucene package includes a module named ExtractReuters which analysts can invoke seamlessly at the command prompt through the Mahout executable. Since Scratch Extractor operates in the same environment (Java Runtime Environment), it too, could execute directly from Mahout once properly integrated with that codebase. The similarity between ExtractReuters and Scratch Extractor extends to the comprising functionality: both aim to transform data serialized in custom formats into simpler inputs for a variety of analytic algorithms. Scratch Extractor, however, preserves more of the underlying structure than does ExtractReuters so that subsequent processing in other modules optionally can leverage this information. One module that does not, Scratch Dispatcher, converts the structured data into its constituent blocks and outputs a comma-separated value (CSV) file useful for importing into distributed file systems, such as Hadoop Distributed File System (HDFS), or graph databases, such as Neo4j, which both promote powerful collaborative filtering techniques. One module that does leverage the structure output by Scratch Extractor, Scratch Traverser, traces the preserved hierarchy while exposing an interface for the analyst to supply custom code executable at each node, thereby enabling functionality limited only by imagination, such as the automated generation of Cypher scripts capable of instantiating Neo4j graph databases. The following sections discuss further details about the implementations of each of Scratch Analyzer's three modules.

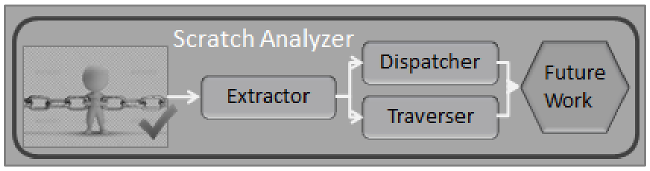
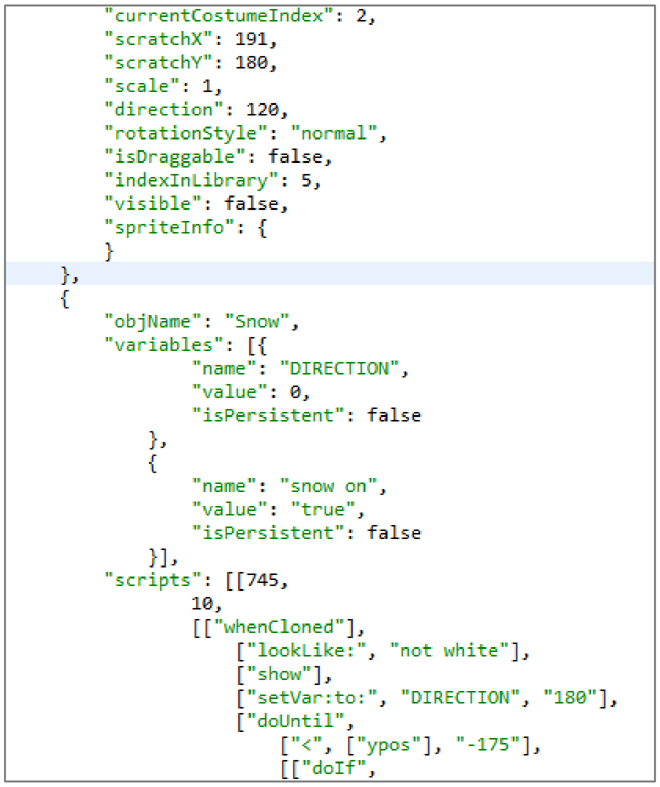


Figure 1. Scratch Analyzer architecture.

B. *Scratch Extractor*

Scratch Extractor aims to adapt the .sb2 file format, designed for saving and loading complete Scratch projects to local file systems, into a more manageable serialization consisting solely of blocks and their associated structural organization of Scratch stacks, known more generally as code blocks. As noted earlier, preserving the hierarchical structure offers implementation flexibility to subsequent Scratch Analyzer modules. However, this preservation does complicate the implementation of the Extractor. A design that merely identifies the blocks used in a project, for example, would involve little more than building a list of all known JavaScript Object Notation (JSON) keywords representative of Scratch blocks, and iterating through the list while employing a search for the existence of each keyword. Although such an implementation might facilitate some big data analytics, it would not provide sufficient extensibility to support the development of solutions to forthcoming and emergent challenges. If one considers Scratch Analyzer a double-sided funnel, Scratch Extractor presents its wide girth in order to secure the vast variety of Scratch projects into a narrow, known, simple format easily digestible by subsequent modules; this stable flow enables, on the others side, an inverse projection outward toward the broad diversity of big data frameworks.

The extraction process first entails decompressing the input set of zipped, custom .sb2 files containing project data and isolating the pertinent JSON representation, such as the example snippet in Figure 2, amidst an assortment of audio and image resources. The second step requires a systematic parsing in order to isolate Scratch blocks from the myriad of other objects serialized, for example, costumes, sounds, and sprites. Next, Scratch Extractor parses the block serialization in order to construct in-memory tree representations for each project. Lastly, a pre-order traversal writes the hierarchical structure into a human-readable format in which tabs represent structural organization, as depicted in the small sample in Figure 3.

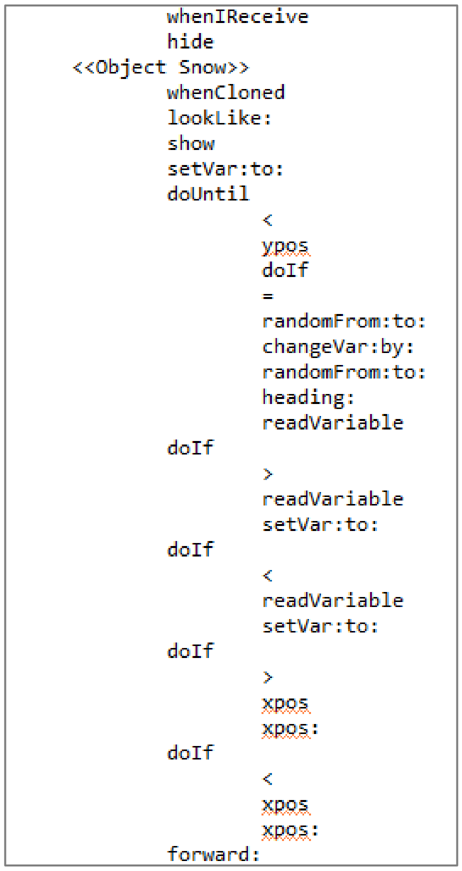


```
        "currentCostumeIndex": 2,
        "scratchX": 191,
        "scratchY": 180,
        "scale": 1,
        "direction": 120,
        "rotationStyle": "normal",
        "isDraggable": false,
        "indexInLibrary": 5,
        "visible": false,
        "spriteInfo": {
        }
    },
    {
        "objName": "Snow",
        "variables": [{
                "name": "DIRECTION",
                "value": 0,
                "isPersistent": false
            },
            {
                "name": "snow on",
                "value": "true",
                "isPersistent": false
            }],
        "scripts": [[745,
                10,
                [["whenCloned"],
                    ["lookLike:", "not white"],
                    ["show"],
                    ["setVar:to:", "DIRECTION", "180"],
                    ["doUntil",
                        ["<", ["ypos"], "-175"],
                        [["doIf",
```

Figure 2. Scratch JSON representation.

C. *Scratch Dispatcher*

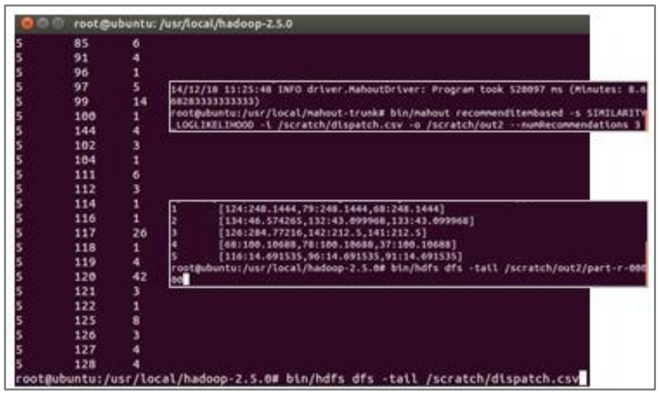
Scratch Dispatcher transforms Scratch blocks contained in multiple .se files output by Scratch Extractor into a set of three CSV files compatible as input sources for a variety of big data platforms. With a target of integrating these block data into collaborative filtering recommendation engines, this process strips the hierarchical structure, reducing the output to a set of simple triplets each consisting of a User ID, a Block, and a Count of the number of times a particular block is used in a set of projects by a particular Scratcher. The files output represent blocks as two different types, string and integer (with an associated lookup dictionary), in order to allow flexibility in subsequent processing; visualizations, for example, might use strings, while recommendation engines frequently require numeric inputs. Unlike content-based recommenders, recommendation engines implementing collaborative filtering do not require extraneous, context-specific information, such as in this case, hierarchical arrangement [59]. The data Scratch Dispatcher aggregates into the simple CSV formats provides sufficient detail to enable the execution of user-based and item-based recommendations in multiple big data environments.



```
    whenIReceive
    hide
<<Object Snow>>
    whenCloned
    lookLike:
    show
    setVar:to:
    doUntil
        <
        ypos
        doIf
        =
        randomFrom:to:
        changeVar:by:
        randomFrom:to:
        heading:
        readVariable
    doIf
        >
        readVariable
        setVar:to:
    doIf
        <
        readVariable
        setVar:to:
    doIf
        >
        xpos
        xpos:
    doIf
        <
        xpos
        xpos:
    forward:
```

Figure 3. Sample Scratch Extractor output.

The dispatch process involves three straightforward steps. First, the Dispatcher constructs per-user lists of in-memory tree representations using the .se files output by Scratch Extractor as inputs. Second, a pre-order traversal of each tree aggregates block counts for each user included in the input set. And third, the Dispatcher serializes the User ID, Block, and Count aggregated into the output CSV files. Preliminary experiments leveraging Scratch Dispatcher's outputs have proven successful. For example, Figure 4 demonstrates the capability to load the output into the map-reduce context of HDFS, run user-based recommendations with Mahout, and discover blocks similar Scratchers have used in their projects. As an example of matching functionality within the graph database paradigm of Neo4j, Figure 5 depicts a Cypher statement that produces parallel recommendation results.

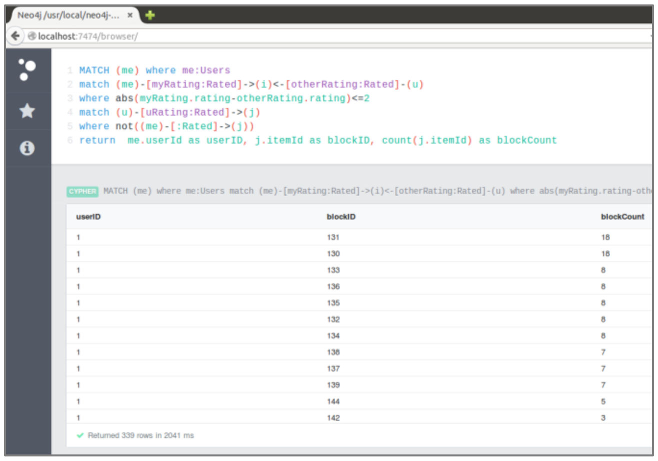


```
root@ubuntu: /usr/local/hadoop-2.5.0
5       85      6
5       91      4
5       96      1
5       97      5
5       99      14
5       100     1
5       144     4
5       102     3
5       104     1
5       111     6
5       112     3
5       114     1
5       116     1
5       117     26
5       118     1
5       119     4
5       120     42
5       121     3
5       122     1
5       125     8
5       126     3
5       127     4
5       128     4
root@ubuntu:/usr/local/hadoop-2.5.0# bin/hdfs dfs -tail /scratch/dispatch.csv
```

```
14/12/18 11:25:48 INFO driver.MahoutDriver: Program took 520097 ms (Minutes: 8.6
68283333333333)
root@ubuntu:/usr/local/mahout-trunk# bin/mahout recommenditembased -s SIMILARITY
_LOGLIKELIHOOD -i /scratch/dispatch.csv -o /scratch/out2 --numRecommendations 3
```

```
1       [124:248.1444,79:248.1444,68:248.1444]
2       [134:46.574265,132:43.099968,133:43.099968]
3       [126:284.77216,142:212.5,141:212.5]
4       [68:100.10688,78:100.10688,37:100.10688]
5       [116:14.691535,96:14.691535,91:14.691535]
root@ubuntu:/usr/local/hadoop-2.5.0# bin/hdfs dfs -tail /scratch/out2/part-r-000
00
```

Figure 4. Scratch blocks loaded into HDFS and recommendations generated via Mahout.



```
MATCH (me) where me:Users
match (me)-[myRating:Rated]->(i)<-[otherRating:Rated]-(u)
where abs(myRating.rating-otherRating.rating)<=2
match (u)-[uRating:Rated]->(j)
where not((me)-[:Rated]->(j))
return  me.userId as userID, j.itemId as blockID, count(j.itemId) as blockCount
```

```
MATCH (me) where me:Users match (me)-[myRating:Rated]->(i)<-[otherRating:Rated]-(u) where abs(myRating.rating-oth
```

userID	blockID	blockCount
1	131	18
1	130	18
1	133	8
1	136	8
1	135	8
1	132	8
1	134	8
1	138	7
1	137	7
1	139	7
1	144	5
1	142	3

✓ Returned 339 rows in 2041 ms

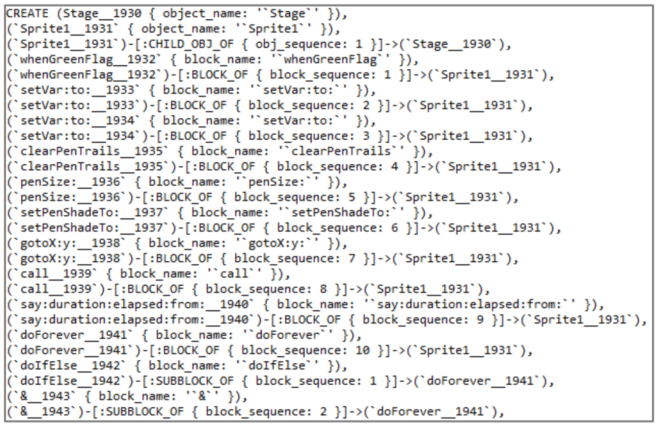
Figure 5. Scratch blocks loaded via Cypher scripts and recommendations generated via Neo4j.

D. *Scratch Traverser*

Scratch Traverser, in contrast to Scratch Dispatcher, fully leverages the hierarchical structure preserved by Scratch Extractor. Rather than merely utilize the structure as a convenient means for accumulating block aggregations, it exposes an interface which facilitates the per-block execution of a user-supplied method during a pre-order traversal. As a consequence, analysts capable of programming in Java or of marshalling matching resources can employ Scratch Traverser as a tool helpful in reaching any end imagined, simply by implementing a class named *Operator* which contains a single static public method named *operate*. This customizability could enable a wide variety of functionality, ranging from electronic tutors which could infer proper block sequences from user-generated solutions with a goal of providing hints to stymied learners [60, 61], to classification algorithms which could automate the determination of whether or not a particular student coherently has used blocks comprising a particular CT concept [62]. Ultimately, the impact of Scratch Traverser depends upon the vision of the analyst.

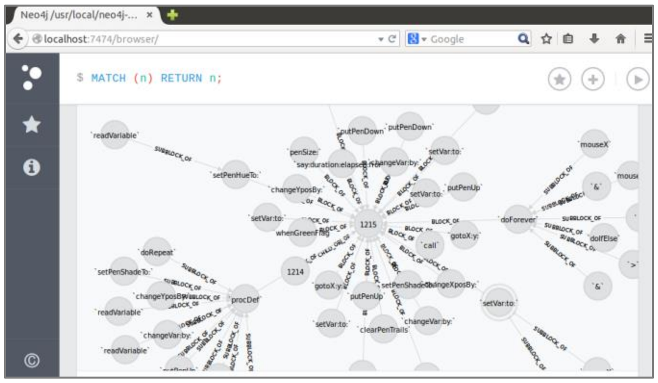
The traversal process first reuses the functionality employed by Scratch Dispatcher to construct per-user lists of in-memory trees from the set of input Scratch Extractor files.

It then performs a pre-order traversal while calling Operate at each node in order to invoke the user-supplied implementation. Scratch Traverser currently includes one example *Operator.operate* implementation which outputs a Cypher script that initializes a Neo4j graph database representing the various arrangements of Scratch blocks used by each Scratcher in each project. Figures 6-7 depict an example snippet of the output Cypher script, and the matching visualization of the graph database once loaded in Neo4j. Having graph database representations of all students' projects could facilitate clustering algorithms which help teachers recognize and isolate learning difficulties in a timelier manner than they might by opening and scrutinizing every Scratch project, one after another. More simply, the visualizations of these graph databases, alone, might prove a motivating force for students eager to expand their graphs and compare them with those of their peers. Scratch Traverser offers analysts, and in classroom practice, educators and students, the capability to pose their own questions about learning, and compose the means to discovering the answers.



```
CREATE (Stage__1930 { object_name: '`Stage`' }),
(`Sprite1__1931` { object_name: '`Sprite1`' }),
(`Sprite1__1931`)-[:CHILD_OBJ_OF { obj_sequence: 1 }]->(`Stage__1930`),
(`whenGreenFlag__1932` { block_name: '`whenGreenFlag`' }),
(`whenGreenFlag__1932`)-[:BLOCK_OF { block_sequence: 1 }]->(`Sprite1__1931`),
(`setVar:to:__1933` { block_name: '`setVar:to:`' }),
(`setVar:to:__1933`)-[:BLOCK_OF { block_sequence: 2 }]->(`Sprite1__1931`),
(`setVar:to:__1934` { block_name: '`setVar:to:`' }),
(`setVar:to:__1934`)-[:BLOCK_OF { block_sequence: 3 }]->(`Sprite1__1931`),
(`clearPenTrails__1935` { block_name: '`clearPenTrails`' }),
(`clearPenTrails__1935`)-[:BLOCK_OF { block_sequence: 4 }]->(`Sprite1__1931`),
(`penSize:__1936` { block_name: '`penSize:`' }),
(`penSize:__1936`)-[:BLOCK_OF { block_sequence: 5 }]->(`Sprite1__1931`),
(`setPenShadeTo:__1937` { block_name: '`setPenShadeTo:`' }),
(`setPenShadeTo:__1937`)-[:BLOCK_OF { block_sequence: 6 }]->(`Sprite1__1931`),
(`gotoX:y:__1938` { block_name: '`gotoX:y:`' }),
(`gotoX:y:__1938`)-[:BLOCK_OF { block_sequence: 7 }]->(`Sprite1__1931`),
(`call__1939` { block_name: '`call`' }),
(`call__1939`)-[:BLOCK_OF { block_sequence: 8 }]->(`Sprite1__1931`),
(`say:duration:elapsed:from:__1940` { block_name: '`say:duration:elapsed:from:`' }),
(`say:duration:elapsed:from:__1940`)-[:BLOCK_OF { block_sequence: 9 }]->(`Sprite1__1931`),
(`doForever__1941` { block_name: '`doForever`' }),
(`doForever__1941`)-[:BLOCK_OF { block_sequence: 10 }]->(`Sprite1__1931`),
(`doIfElse__1942` { block_name: '`doIfElse`' }),
(`doIfElse__1942`)-[:SUBBLOCK_OF { block_sequence: 1 }]->(`doForever__1941`),
(`&__1943` { block_name: '`&`' }),
(`&__1943`)-[:SUBBLOCK_OF { block_sequence: 2 }]->(`doForever__1941`),
```

Figure 6. Scratch Traverser output: Cypher script snippet.



```
$ MATCH (n) RETURN n;
```

Figure 7. Scratch block graph database visualized in Neo4j.

IV. Future Work

"We shouldn't "teacher-proof" curricula so that it can be tested; instead, we should create compelling materials that address teachers' needs and inspire them to teach creatively and effectively" [4].

The motivation to construct Scratch Analyzer derives from CS doctoral research entitled SAGE, which aims to develop, deploy, and evaluate a collaborative game-based learning system which infuses computational thinking within K-12 curricula. Scratch, as SAGE's underlying platform, offers robust connections to a network of collaborative communities, as well as principled dedication to constructionism, the learning theory which encourages the cultivation of understandings through the creation of personally meaningful artifacts. Since key features of Scratch enable the remixing of projects others have shared [46], analytical advances such as the collaborative filtering herein discussed could empower learners, with ease, to discover peers and projects relevant to them and their learning goals, thereby positively impacting motivation, persistence, and outcomes. Educators, meanwhile, could benefit from the affordances of clustering and classification algorithms which could identify gaps in group and individual learning trajectories, and inform the student evaluation process.

The work completed in Scratch Analyzer adeptly interconnects Scratch projects to big data infrastructures. To manifest change in informal and formal learning environments, though, educators and learners likely will need more support than currently provided. As the development of SAGE unfolds in the coming years, Scratch Analyzer should help facilitate the incorporation of the aforementioned EDM and LA techniques capable of adaptively customizing and personalizing learning. User-based and item-based recommenders could integrate within the learning system so that accessibility reduces to a few clicks, or none at all, in an environment in which guidance auto-generates as a result of student struggles recognized via auto-analysis. These real-time suggestions could enable students to discover peers who have worked through similar problems recently, thereby enabling children to learn from peers who have just developed the same conceptual understandings, rather than from educators expert in content areas who often misconstrue adequate sequencing in their instruction for novices [63]. The connectivity to peers recently reaching the horizon of conceptual understanding could produce the dual impact of reinforcing the newly grasped clarity for the slightly more advanced students as they explain to less proficient learners, who then eclipse the cusp of comprehension. Such a peer-discovery system, alone, could measurably enhance learning outcomes, but Scratch Analyzer should enable far more than just recommendations. The architecture of Scratch Traverser, in particular, could bind to a variety of user-supplied implementations with multiplicities of aims. To encourage its usage, however, SAGE likely will need to cultivate a user-friendly ecosystem for Scratch Traverser

Operator.operate implementations so that educators less sophisticated, or without sufficient time or resources, can easily adopt or adapt *operate* implementations from the community. Future work, in other words, entails continuing to expand functionality while adding a streamlined user interface in order to simplify usage in practice in learning environments. Scratch, with Scratch Analyzer atop, establishes a steady platform; SAGE intends to leverage the platform by blending these capabilities seamlessly into its collaborative GBL system infusing CT into K-12 curricula.

V. CONCLUSIONS

"As EDM and LA become used in a wider variety of domains, by researchers from a wider variety of disciplines, and within learning systems of a wider variety of types, we will see the potential of these approaches for enhancing both practice and theory in the learning sciences" [13].

To summarize, for learning-system designers who aim to infuse computational thinking within K-12 curricula, Scratch Analyzer is an open-source software suite that distills Scratch projects into inputs fit for insightful EDM and LA. Architected modularly in order to maximize flexibility and extensibility, the system consists of three components: Scratch Extractor, Scratch Dispatcher, and Scratch Traverser. Scratch Extractor identifies Scratch blocks serialized as JSON and strips irrelevant syntax while preserving hierarchical structure. Scratch Dispatcher transforms extracted blocks into simple CSVs compatible with recommendation engines built using a variety of big data technologies. Scratch Traverser follows the hierarchical structure of extracted Scratch blocks and executes a user-supplied method for each block. In combination, these components enable Scratch Analyzer to automate time-consuming instructional and learning responsibilities by offering teachers, and their students, opportunities to access timely formative and summative feedback as they strive together toward mastery.

With platforms such as Scratch, and toolkits such as Scratch Analyzer, a bright future awaits children entering pre-school during the birth of the big data era. By assimilating the data exhausted during interactive learning, dynamic educational systems can properly pace, customize, adapt, and personalize learning situated within intrinsically motivating GBL environments. CT, once hardly acknowledge as broadly valuable, can become pervasively invaluable, thereby empowering the general public to leverage a computational mindset when encountering the big data increasingly permeating daily lives. This transformational potential appears poised to contribute to a revolution in education, schooling, and life-long, life-wide, life-deep learning.

ACKNOWLEDGMENT

Thank you to Computer Science Professor Gail Kaiser and Teachers College Professor Joey Lee for advising the SAGE research agenda. Additional thanks to TA Huan Gao for his responsive assistance throughout the semester, and Professor Lin for offering the practical and enjoyable class.

REFERENCES

[1] W-H. Wu, H-C. Hsiao, P-L Wu, C-H. Lin, and S-H. Huang, "Investigating the learning-theory foundations of game-based learning: a meta-analysis," Journal of Computer Assisted Learning. 28, 2012, 265–279.

[2] D. S-C. Dalmau, Core Techniques and Algorithms in Game Programming. New Riders Publishing, U.S., 2003.

[3] J. M. Randel, B. A. Morris, C. D. Wetzle, And B. V. Whitehead, "The effectiveness of games for educational purposes: A review of recent research," Simulation and Gaming, 23, 1992, 261–276.

[4] K. Squire, Video Games and Learning: Teaching and Participatory Culture in the Digital Age. Teachers College Press, New York, 2011.

[5] R. Schank, Teaching Minds: How Cognitive Science Can Save Our Schools. Teachers College Press, New York, 2011.

[6] H. Jenkins, Convergence Culture: Where Old and New Media Collide. NYU Press, New York, 2006.

[7] IBISWorld, Movie & Video Production in the US. Industry Report 51211a, 2013.

[8] IBISWorld. Music Publishing in the US. Industry Report 51223, 2013.

[9] IBISWorld. Video Games in the US. Industry Report NN003, 2013.

[10] C. Wilson, L. A. Sudol, C. Stephenson, and M. Stehlik, "Running on empty: The failure to teach K-12 computer science in the digital age," ACM, 2010. Retrieved November 29, 2014 from http://www.acm.org/runningonempty/fullreport2.pdf.

[11] R. Florida, The Rise of the Creative Class. Basic Books, 2002.

[12] M. Resnick, "Computer as paint brush: Technology, play, and the creative society," in Play= Learning: How Play Motivates and Enhances Children's Cognitive and Social-Emotional Growth. Oxford University Press, 2006.

[13] R. Baker and G. Siemens, "Educational data mining and learning analytics," in The Cambridge Handbook of the Learning Sciences, K. Sawyer, (Ed.). Cambridge University Press, 2014.

[14] Y. B. Kafai, K. A. Peppler, and R. N. Chapman, The Computer Clubhouse: Constructionism and Creativity in Youth Communities. Teachers College Press, 2009.

[15] B. J. Reiser and I. Tabak, "Scaffolding," in The Cambridge Handbook of the Learning Sciences, K. Sawyer, (Ed.). Cambridge University Press, 2014.

[16] T. Wagner, The Global Achievement Gap: Why Even Our Best Schools Don't Teach the New Survival Skills Our Children Need – and What We Can Do. Basic Books, 2008.

[17] S. Khan, The One World Schoolhouse: Education Reimagined. Twelve, 2012.

[18] T. Wagner, Creating Innovators: The Making of Young People Who Will Change the World. Scribner, 2012.

[19] W. Bainbridge, "Carnegie unit," in Encyclopedia of Educational Reform and Dissent, T. Hunt, J. Carper, T. Lasley, and C. Raisch, (Eds.). SAGE Publications, Inc., Thousand Oaks, CA, 2010, 136-138.

[20] T. Guskey And S. Gates, "Synthesis of research on the effects of mastery learning in elementary and secondary classrooms," Educational Leadership, 43, 8, 1986.

[21] D. Ravitch, The Death and Life of the Great American School System: How Testing and Choice Are Undermining Education. Basic Books, 2011.

[22] D. L. Mitra, “The significance of students: Can increasing "student voice" in schools lead to gains in youth development?,” Teachers College Record, 106, 4, 2004, 651–688.

[23] U. Mager And P. Nowak, “Effects of student participation in decision making at school: A systematic review and synthesis of empirical research,” Educational Research Review, 7, 1, 2012, 38–61.

[24] D. T. Willingham, Why Don't Students Like School: A Cognitive Scientist Answers Questions About How the Mind Works and What It Means for the Classroom. Jossey-Bass, 2009.

[25] M. Prensky, Don't Bother Me Mom – I'm Learning! Paragon House, 2006.

[26] J. McGonigal, Reality Is Broken: Why Games Make Us Better and How They Can Change the World. Penguin Press, 2011.

[27] K. Squire, “Replaying history: Learning world history through playing civilization III,” unpublished Ph. D. dissertation. Indiana University, Indiana, 2004.

[28] J.P. Gee, What Video Games Have To Teach Us About Learning. Palgrave, 2003.

[29] K. K. Szpunar, N. Y. Khan, and D. L. Schacter, “Interpolated memory tests reduce mind wandering and improve learning of online lectures,” Proceedings of the National Academy of Sciences USA, 110, 16, 2013.

[30] W. B. Wood and K. D. Tanner, “The role of the lecturer as tutor: Doing what effective tutors do in a large lecture class,” Life Sciences Education, 11, 3, 2012.

[31] R. Garris, R. Ahlers,, and J. E. Driskell, “Games, motivation, and learning: A research and practice model,” Simulation & Gaming, 33, 4, 2002, 441–467.

[32] R. Ryan, C. Rigby, and A. Przybylski, “The motivational pull of video games: A self-determination theory approach,” Motivation and Emotion, 30, 4, 2006.

[33] Y. B. Kafai, K. A. Peppler, M. Resnick, and N. Rusk, “Programming by choice: Urban youth learning programming with scratch,” ACM SIGCSE Bulletin, 40, 1, 2008.

[34] E. R. Halverson, “Participatory media spaces: A design perspective on learning with media and technology in the twenty-first century,” in Games, Learning, and Society (Learning in Doing: Social, Cognitive and Computational Perspectives), C. Steinkuehler, K. Squire, and S. Barab, (Eds.). Cambridge University Press, 2012.

[35] S. De Freitas, F. Liarokapis, “Serious games: A new paradigm for education?,” in Serious Games and Edutainment Applications, M. Ma, A. Oikonomou, and L. Joain, (Eds.), Springer, 2011.

[36] M. Csíkszentmihályi, Flow: The Psychology of Optimal Experience. Harper Perennial, 1990.

[37] C. Bachen and C. Raphael, “Social flow and learning in digital games: A conceptual model and research agenda,” in Serious Games and Edutainment Applications, M. Ma, A. Oikonomou, and L. Joain, (Eds.), Springer, 2011.

[38] J.P. Gee and E. Hayes, Nurturing affinity spaces and game-based learning,” in Games, Learning, and Society (Learning in Doing: Social, Cognitive and Computational Perspectives), C. Steinkuehler, K. Squire, and S. Barab, (Eds.). Cambridge University Press, 2012.

[39] J. Wing, “Computational thinking and thinking about computing,” Philosophical Transactions: Mathematical, Physical and Engineering Sciences, 366, 1881, 2008.

[40] J. Wing, “Computational thinking,” Communications of the ACM, 49, 3, 2006, 33-35.

[41] Committee For The Workshops On Computational Thinking, “Report of a workshop on the scope and nature of computational thinking,” National Academies Press, Washington, D.C., 2010.

[42] Computer Science Teachers Association, “Operational definition of computational thinking for K-12 education, CSTA, 2011. Retrieved November 18, 2014 from http://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf.

[43] L. A. Gouws, K. Bradshaw and P. Wentworth “Computational thinking in educational activities: An evaluation of the educational game light-bot,” Proceedings of the 18th ACM conference on Innovation and technology in computer science education, ACM, 2013.

[44] M. Prensky, “Digital natives, digital immigrants part 1,” On the Horizon, 9, 5, 2001, 1-6.

[45] A. A. diSessa, Changing Minds: Computers, Learning, and Literacy. MIT Press, 2000.

[46] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, E. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, “Scratch: Programming for all,” Communications of the ACM 52, 11, 2009, 60-67.

[47] M. Resnick and B. Silverman, “Some reflections on designing construction kits for kids,” Proceedings of International Conference for Interaction Design and Children, 2005.

[48] A. E. Weinberg, “Computational thinking: An investigation of the existing scholarship and research,” unpublished Ph. D. dissertation. Colorado State University, Colorado, 2013.

[49] C. Hu, “Computational thinking – what it might mean and what we might do about it,” Proceedings of the 16th annual joint conference on innovation and technology in computer science education, ACM, 2011.

[50] I. Lee, F. Martin, J. Denner, B. Coulter, W. Allan, J. Erickson, J. Malyn-Smith, and L. Werner, “Computational thinking for youth in practice,” ACM Inroads, 2, 1, 2011, 32-37.

[51] P. Sengupta, J. Kinnebrew, S. Basu, G. Biswas, and D. Clark, “Integrating computational thinking with k-12 science education using agent-based computation: A theoretical framework,” Educational Information Technology, 18, 2, 2013, 351-380.

[52] U. Wilensky and M. J. Jacobson, “Complex systems and the learning sciences,” in The Cambridge Handbook of the Learning Sciences, K. Sawyer, (Ed.). Cambridge University Press, 2014.

[53] A. Basawapatna, A. Repenning, And C. Lewis, “The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking,” Proceeding of the 44th ACM technical symposium on Computer science education, ACM, 2013.

[54] A. Yadav and J. Korb, “Learning to teach computer science: The need for a methods course,” Communications of the ACM, 55, 11, 2012.

[55] A. Yadav, N. Zhou, C. Mayfield, S. Hambrusch, and J. Korb, “Introducing computational thinking in education courses,” Proceedings of the 42nd ACM technical symposium on Computer science education, ACM, 2011.

[56] C. Dierbach, H. Hochheiser, S. Collins, G. Jerome, C. Ariza, T. Kelleher, W. Kleinsasser, J. Dehlinger, and S. Kaza, “A model for piloting pathways for computational thinking in a general education curriculum,” Proceedings of the 42nd ACM technical symposium on computer science education, ACM, 2011.

[57] D. Loshin, Big Data Analytics: From Strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph. Morgan Kaufmann, 2013.

[58] A. Nugent, F. Halper, M. Kaufman, Big Data for Dummies. For Dummies, 2013.

[59] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action. Manning Publications, 2011.

[60] J. Stamper, T. Barnes, and M. Croy, “Extracting student models for intelligent tutoring systems,” Proceedings of the 22nd national conference on artificial intelligence, volume 2, AAAI Press, 2007, 1900-1901.

[61] J. Stamper, T. Barnes, and M. Croy, “Enhancing the automatic generation of hints with expert seeding,” Proceeding of the 10th International Conference on Intelligent Tutoring Systems, volume 2, 2010, 31-40.

[62] U. Wolz, C. Hallberg, and B. Taylor, “Scrape: A tool for visualizing the code of Scratch program,” Poster presented at the 42nd ACM Technical Symposium on Computer Science Education, 2011.

[63] C. Christensen, and C. W. Johnson, Disrupting Class, Expanded Edition: How Disruptive Innovation Will Change The Way The World Learns. McGraw-Hill, 2010.