SAGE-RA: A Reference Architecture to Advance the Teaching and Learning of Computational Thinking

Johan Sulaiman[1], Alex Dziena[2], Jeff Bender[3], and Gail Kaiser[4]

[1]Columbia University, johan.sulaiman@columbia.edu
[2]Columbia University, ad3363@columbia.edu
[3]Columbia University, jeffrey.bender@columbia.edu
[4]Columbia University, kaiser@cs.columbia.edu

Abstract

Rapid technological advances and the increasing number of students in Southeast Asian nations present a difficult challenge: how should schools adequately equip teachers with the right tools to effectively teach Computational Thinking, when the demand for such teachers outstrips their readiness and availability? To address this challenge, we present the SAGE reference architecture; an architecture for a learning environment for elementary, middle-school and high-school students based on the Scratch programming language. We synthesize research in the domains of game-based learning, implicit assessments, intelligent tutoring systems, and learning conditions, and suggest a teacher-assisting instructional platform that provides automated and personalized machine learning recommendations to students as they learn Computational Thinking. We discuss the uses and components of this system that collects, categorizes, structures, and refines data generated from students' and teachers' interactions, which also facilitates personalized student learning through: 1) predictions of students' distinct programming behaviors via employment of clustering and classification models, 2) automation of aspects of formative assessment formulations and just-in- time feedback delivery, and 3) utilization of item-based and user-based collaborative filtering to suggest customized learning paths. The proposed reference architecture consists of several architectural components, with explanations on their necessity and interactions to foster future replications or adaptations in similar educational contexts.

Index Terms: *Computational Thinking; Machine Learning, Game-Based Learning; Implicit Assessment; Intelligent Tutoring Systems*

Rationale and Objectives

Jeannette Wing, in her influential article "Computational Thinking" (Wing, 2006), recognizes that the jobs of the future will increasingly rely on the aid of programmable machines to meet higher standards of acceptable productivity in all industries and workplaces. In the United States alone, the number of information technology workers stood at 4.6 million in the year 2016 compared to just 450,000 in 1970, and computer and mathematical occupations will continue to rise by 18 percent between 2012 and 2022 (United States Census Bureau, 2016). Workers who seek to maximize their output through machine-based and computer-based enablers are required to master multi-layered concepts and mental competencies such as

abstraction, decomposition, recursion, iteration, modeling, and many others. These myriad areas of Computational Thinking (CT) are complex concepts that are necessarily abstract, in order for them to be applicable to virtually every industry that can leverage machine computations to augment human workers. This translates to a steep and extensive learning curve for students who aspire to master CT, requiring persistence, creativity, and mental resiliency (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986).

The abstract and complex nature of this domain presents at least two major challenges. First, the daunting learning curve makes it difficult to spark students' interest in undertaking CT learning, let alone harnessing and sustaining that motivation through the learning journey. Second, even if students' manage to overcome this hurdle, there is a shortage of teachers who have been adequately trained to be proficient in CT, and are adept in teaching the subject to students. The drought of qualified teachers is felt in many parts of the world; in the United States, only one-third of Computer Science high school teachers in the United States has a computer science degree, and it is estimated that there will be an unmet need of more than 20,000 computer science teachers in the next decade (Leyzberg & Moretti, 2017). Similarly, a report by Southeast Asian Ministers of Education Organization (SEAMEO) highlights a shortage of qualified teachers in Southeast Asian countries such as Indonesia, Thailand, and Singapore, recommending development of practical mass customization educational model that adapts education routes to cater to students' different needs, abilities, aptitudes and learning modalities (Sadiman, 2004).

Social Addictive Gameful Engineering (SAGE), a learning platform that infuses CT and gamification to bring computational skills to elementary, middle-school and high school students, seeks to answer these challenges by increasing students' interests through practical presentation of CT concepts in a gameful way and by assisting teachers in providing guidance, hints, evaluation, and progress feedback on students' learning progress through an Intelligent Tutoring System (ITS).

The value of reference architectures during the development of software systems is well documented (Angelov, Grefen, & Greefhorst, 2009). In building the SAGE ITS, we researched, optimized, and validated the SAGE Reference Architecture (SAGE-RA), which consists of interlocking components that work together in providing: 1) student programming behavior prediction, 2) formative assessment formulation and formative feedback delivery, and 3) learning path personalization.

SAGE-RA is intended to provide a basis for members of the research community to replicate and/or adapt the work done on SAGE and validate their own work related to learning systems against a standard architecture. We identify several crucial components in a learning system and their interactions with one another, which can aid researchers in selecting system components among a vast universe of potential choices or validate their own selections, and provide guidance for the design of fundamental component interactions in such a learning system. Further, the reference architecture provides a potential basis for the standardization of interaction protocols, which would enable multiple independent researchers to focus on studying and developing the details of particular components, and enable the community to more easily utilize components that conform to such a standard protocol.

Perspectives and theoretical framework

Scratch

One example of a widely adopted open-ended learning environment is Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Scratch is a block-based programming environment inspired by the LOGO programming language, where children are encouraged to create programming projects using block objects similar to LEGO blocks. Scratch can be used to create games and interactive media in ways that are welcoming to younger age audiences new to programming. Scratch provides the unrestrictive open-ended game design space that students respond well to (Resnick et al., 2009), allowing for self-paced explorations of virtual blocks that when linked correctly become executable programs that come alive and shareable with Scratch's active multi-million user-base. In an endeavor to learn programming, novice students typically would benefit significantly from guidance from domain experts who are capable and effective in imparting knowledge (Perkins et al., 1986). The challenge is how to make these interactions traditionally seen in physical and organized classroom settings translatable in the context of an online learning environment. The SAGE architecture leverages Scratch's play-like element as a learning medium to impart CT knowledge while allowing teachers to embed goal-based learning path and assignments (called 'quests' and 'games') and facilitates the utilization of an automated responsive feedback mechanism that mimics the experience of having a knowledgable teacher or tutor sitting alongside the student and providing guidance, in a way that is immersive.

Implicit Assessments

While gamification is useful in encouraging and keeping students' motivation with the hope of prolonged engagement in the learning platform and fostering learning in the process, more empirical evidence and curriculum-specific learning outcomes are needed to conclusively prove that games are an educationally effective solution (Kazimoglu, Bacon, & Kiernan, 2010). Devising an effective assessment strategy in measuring student's learning progress while using construction-oriented games thus is an area of importance. Frameworks such as evidence-centered design (ECD) are attractive (Mislevy & Haertel, 2006), because they aim to structure learners' performance data that are continuously collected from the game, utilizing machine-based reasoning techniques to make inferences about the learner's competencies resulting in a learner model (Rupp, Gushta, Mislevy, & Shaffer, 2010). It is argued that in-session assessment and ECD can assess not only content-specific learning but also general abilities, dispositions and beliefs, thus being more adequate to assess the so-called 21st century competences such as problem-solving skills, persistence, and creativity (Carvalho, 2017). This design philosophy influences SAGE, and is reflected in the SAGE architecture and modules. The gameful approach of the SAGE user experience in the front-end, for example, also serves functions that are similar to the Presentation Model in the ECD framework, where hints are delivered after being evaluated, customized, and constructed in real-time.

Intelligent Tutoring Systems

Students who embark in their learning journey with the benefit of a tutor's close involvement arguably are on the most efficient path of acquiring domain-specific skill and knowledge (VanLehn, 2006). There has been efforts in trying to replicate the effective curriculum design, hinting, and guidance of a teacher as part of the output of ITS, and in understanding these actions it is helpful to study these systems' behavioral distinctions from two vantage points: the outer loop and the inner loop. The outer loop is primarily concerned about the

logical and scaffolded sequencing between learning tasks, similar to how course topics are arranged in sequence to minimize difficult cognitive jumps the students have to perform. An ITS inner loop on the other hand, is one level deeper in granularity compared to the outer loop, where the ITS is more concerned in helping induce the self-recognition of the student in understanding the current step or place the student is in within a task, and in taking the action of selecting and acting on the next correct step that will lead the student closer towards completing the learning task at hand. These conceptions of outer and inner loops provide a basis for the implementation of learning path recommendation and real-time hint generation in the SAGE Reference Architecture.

Conditions of Learning in Novice Programmers

During an analysis of elementary and high-school students' learning patterns as they get familiar with BASIC and LOGO programming (Perkins et al., 1986), it appears that students' common programming behaviors could be categorized into four learner types: Stoppers, Movers, Extreme Movers, and Tinkerers. Students of the Stopper type, for example, tend to stop any further exploration of problem solving at early encounters with obstacles, although it is important to note that the right and timely intervention from a guide has shown some success in encouraging more exploration, even for Stoppers. The Mover type is the ideal learner type, yielding the most optimal learning rate, where students exhibit continuous activities and prolonged interaction with the learning system. However moving at an exceedingly fast rate with more randomness of movement is more indicative of the Extreme Mover learner type. This type is less desirable because it seems to indicate the student's emotional detachment from the task. There is yet another learner type, called Tinkerers, that is a subset of the Mover type. This type is distinctive in its tendency to evaluate its latest action and make micro modifications in an attempt to stumble into or arrive at the desired outcome, depending on the student's grasp of the syntax and learning concept at hand. Taken as a whole, this research on the different learner types provides the programming behavior classifications useful for SAGE's machine learning models in providing just-in-time hints and guidance during students' learning experience.

The Gameful Computational Thinking Education Domain

We define the Gameful Computational Thinking Education Domain (GCTED) as the domain of pedagogical systems that apply Game Based Learning (GBL) to enable students to learn CT. In particular, GCTED systems aim to enable learning in students that have previously had limited or no exposure to CT concepts. GBL has been explored as a pedagogical tool since the early 1960's (J. M. Randel, 1992), and Gee's 2003 treatise (Gee, 2003) highlighting it's efficacy, as well as studies demonstrating it's successful use in classrooms (Squire, 2004), have spurred greatly renewed interest in GBL. Two game design frameworks within the larger set of existing game design patterns (C. Harteveld, 2014) deserve special focus in the context of GCTED: Parson's Programming Puzzles (PPPs) (Parsons & Haden, 2006) and Constructionist Video Games (CVGs) (D. Weintrop, 2012).

Using PPPs, a teacher can impart good programming practices via direct instruction, enabling students to practice CT by reordering blocks of well-written code which focus on individual concepts into a functioning program (Kim & Axelrod, 2005). Block-based sorting algorithm puzzles, in which students reorder blocks of code to produce a functioning sorting algorithm, are examples of PPPs that highlight the efficacy of the framework, particularly in the context of puzzles of relatively small scope. PPPs reduce grading effort and variability (Cheng & Harrington, 2017), improve the efficiency of learning (B. J. Ericson, 2017), and allow formative

feedback to activate when needed (J. Helminen, 2012).

CVGs provide expressive design tools that allow personally meaningful gameplay. Teachers define incremental goals (Reiser & Tabak, 2014), and students take ownership of the experience, stimulating interest and increasing motivation, leading to sustained engagement (J ärvelä & Renninger, 2014).

These frameworks can be used in conjunction; when students engaging in PPPs show evidence of comprehension, CVGs could enhance PPPs' impact by providing a new context in which students can apply CT concepts, and thus improve students' understanding.

Methods

We used two frameworks for the derivation (Hassan & Holt, 2000) and classification (Angelov et al., 2009) of a new reference architecture, useful in developing constructionist and direct instruction learning environments within the GCTED. We then validated the proposed architecture against another web-based learning environment that is widely used, Code.org®.

As we had implemented much of the SAGE ITS before creating a reference architecture, and because several other widely used applications within the emergent GCTED domain had similarly been created without a reference architecture, our primary criteria for selection of a reference architecture derivation framework were its applicability to existing concrete architectures, and its applicability within emergent domains. Hassan and Holt's framework was selected because of its demonstrated applicability to derive a reference architecture from an existing implementation through reverse engineering (Hassan & Holt, 2000); specifically the authors derive a reference architecture for web servers, an emergent domain at the time the study was published, from three existing implementations. The reverse engineering approach detailed by Hassan and Holt also provided us the method through which we validated SAGE-RA: mapping the reference architecture to components of existing concrete implementations. In this paper, we share the details of one of the mappings from SAGE-RA to a concrete architecture, SAGE-RA to Code.org®, by reverse engineering conceptual architectures for that implementation.

Classification of SAGE-RA provided guidance in the design of an effective architecture. Being able to successfully classify the architecture also partially validated its general applicability throughout a domain. Angelov e.a's classification framework (Angelov et al., 2009) was selected because of its applicability to a wide range of architecture types, including architectures within emergent domains such as GCTED.

Deriving SAGE-RA

Using a process similar to Hassan and Holt's (Hassan & Holt, 2000), we first derived a concrete architecture for SAGE based on the existing implementation. We then derived a conceptual architecture, based on a generalization of the concrete architecture and plans for future development. From the conceptual architecture, we proposed a reference architecture, validated it against another implementation in the domain, and refined it.

Validating SAGE-RA via Mapping

Code.org® is a nonprofit dedicated to expanding access to computer science in schools and increasing participation by women and underrepresented minorities. Code.org® provides the leading curriculum for K-12 computer science in the largest school districts in the United States and Code.org® also organizes the annual Hour of Code campaign which has engaged 10% of all

students in the world (About Us — Code.org, n.d.). We used Code.org®'s GitHub repository (code-dot org, 2019) to map the primary components of their application to components of SAGE-RA.

Results and Conclusion

We identified SAGE's architecture as a hybrid of type 3.1 and type 5.1 per Angelov e. a.'s framework (Angelov et al., 2009): a preliminary, classical, facilitation reference architecture for multiple organizations.

SAGE's Concrete Architecture

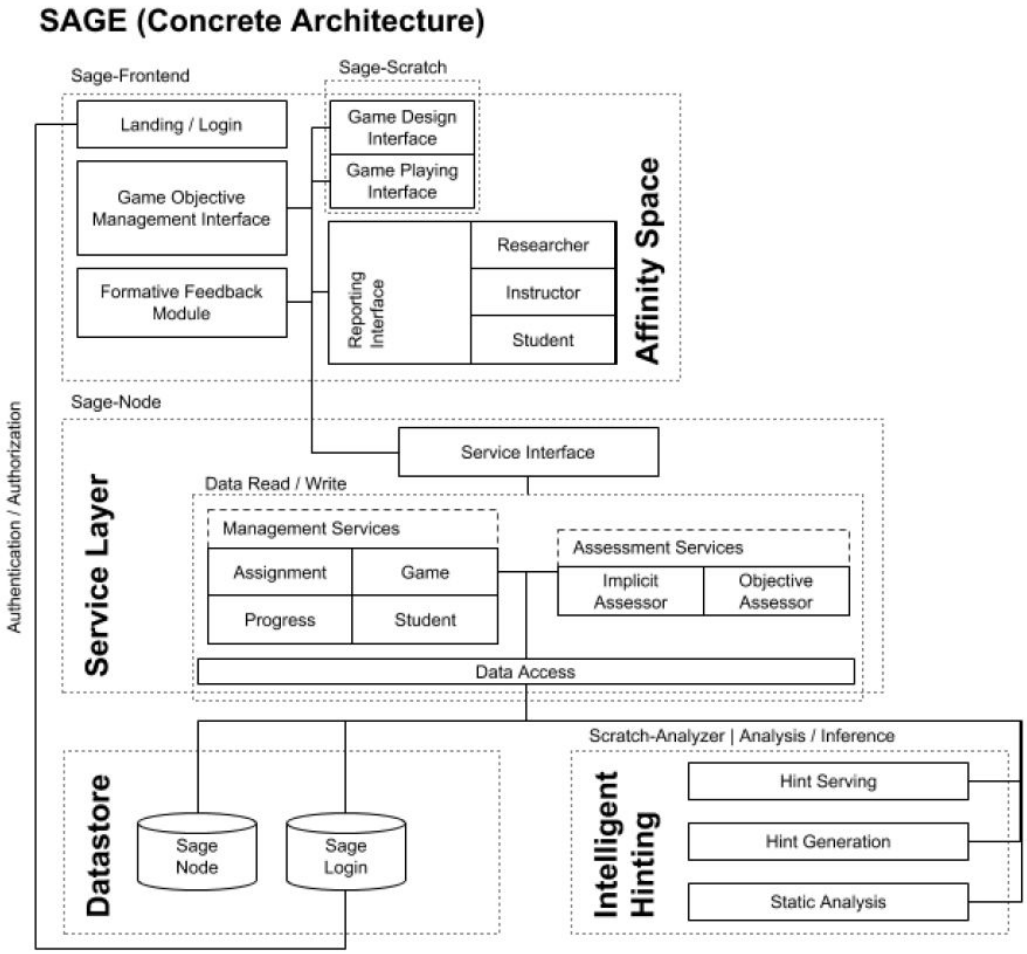

Fig. 1. SAGE Concrete Architecture

The below is a brief walkthrough of the different components, groupings and modules within the SAGE Concrete Architecture:

1) SAGE Affinity Space: The Affinity Space, primarily through the Sage-Frontend grouping, is the presentation component of SAGE, implemented as a Node.js app using AngularJS, Bootstrap, and JQuery, that handles user management functions such as logins, registrations, and profile management. In addition the Affinity Space also displays learning path presentations, and provides formative assessment through the Formative Feedback module.

Outer-loop Machine Learning recommendations are being served in these modules. When a student launches a learning session from the Game Objective Management Interface, the session occurs from within the Sage-Scratch grouping. The Affinity Space also supplies context-specific reporting for the three types of users: Researcher, Instructor, and Student. Moreover, from the end user perspective, the Affinity Space also functions as the interactive content distribution and engagement platform for these various types of users to share and consume other users' various types of content, in effect providing a virtual community of learners. Teachers may find the growing content, the archive of interactions, and the feedback-giving and feedback-receiving mechanism available through the Affinity Space useful in providing effective teaching.

2) SAGE Service Layer: The Service Layer is the back-end service component for SAGE, that handles and completes various requests from the Affinity Space. The Management Services grouping within this Service Layer processes creation, update, and deletion of assets related to Assignment, Game, Progress, and Student in accordance to their respective associated built-in business rules. The Assessment Services grouping handles the serving of implicit and objective assessments of the student's performance towards achievement of learning goals, either via teacher-defined built-in rules, or via automated machine-learning generated results received from the Intelligent Hinting component.

3) SAGE Intelligent Hinting Layer: The Intelligent Hinting component is capable to formulate AI generated Assessment and Formative Feedback (Piech, Sahami, Huang, & Guibas, 2015). This includes assessment of progress attainment towards game objectives and learning paths (Owen, Shapiro, & Halverson, 2013), which is fed back in the form of in-session messages, suggestions, and hints. As in-game assessment has been shown to improve learning outcomes for novice-programmers (Lee, Ko, & Kwan, 2013) a flexible assessment model is necessary to insure that a SAGE-RA compliant implementation can use a wide array of feedback and assessment models, in order to keep up with continuing research on efficacy of feedback types in game-based learning.

To source the student interaction data from the Affinity Space presentation layer, the Sage-scratch grouping in that layer takes a game snapshot, represented in a JSON file, and sends it to Sage-node in the Service layer to be saved as a field to the database in the Datastore component. Each individual file contains rich transactional and referential information such as timestamp, game object attributes (unique identifier, type, position, and coordinate), progress markers, and student association. The file is then used within the Scratch-Analyzer grouping where it undergoes predefined data wrangling and data cleansing processes, and becomes the basis for statistical data analysis and machine learning processing. State transitions throughout the session are computed by tracking deltas between snapshots.

Machine learning recommendations in SAGE are provided to the outer loop and the inner loop. In the SAGE concrete implementation, the outer loop is the process that runs when the students complete games and quests that serve as building blocks that can be linked and arranged cohesively together to form a customized learning path adapted to the needs of each individual student. to improve skills where they are struggling, and find new challenges as they master current games. To formulate this personalized game and quest structure, SAGE operationalizes a recommendation engine that employs item-based and user-based collaborative filtering system that arrange in sequence the set of quests and games the student can pursue individually.

In the student case, we analyze pairwise similarity between students across the set of games

that each pair has played. Factors in the comparison include game difficulty, game score, game-objective score, and static analysis scores for seven CT concepts: Abstraction, Parallelization, Logic, Synchronization, Flow Control, User Interactivity, and Data Representation. From each pair of <student1, student2>, we compute student 1's highest score across games for each of the above CT concepts. If student 1 expresses a need for additional practice, we activate a struggling algorithm, which identifies both CT concepts in which student 1 has low mastery, and the games that focus on those CT concepts above a configured proportion of all CT concepts, with the proportion assigned as the similarity score for that game between <student1, student2>. This allows a student to select games that are optimized for that student.

For teachers, we construct an average-based model of a student in the teacher's class, which includes an average score and the average mastery of each CT concept of all students in the class. We then score all other games available, via the struggling algorithm and the mastery similarity calculation, and recommend the highest scored games to the teacher for assignment to the class. This allows the teacher to optimize game selection.

SAGE's inner loop, on the other hand, employs the use of a Visual Assessment Editor, a block-based language and library that deconstructs the syntax and types of Instructor hints and guidance to the students. Inner loop algorithms are provided in the specific terms of the Visual Assessment's unique vocabulary and grammar. In the concrete architecture, real-time assessments are configurable to trigger at the teacher's desired interval when the student is in game, or automatically when specific actions are made by the student.

Student gameplay data is used to generate Decision Tree and K Nearest Neighbors classification models, which outperformed other algorithms on mock data, based on the time-series representation of block placement in the scripts pane, allowing us to classify programming behavior as one of the four learner types discussed above: Stoppers, Movers, Extreme Movers, and Tinkerers.

This detected programming behavior is used as one input into our mechanism for providing intelligent hinting to students potentially in need of individualized assistance during gameplay; other inputs include unsupervised identification of game milestones and gameplay paths through those milestones, supervised classification of students by path and programming behavior, and the generation of a Poisson Path to help determine the shortest path for a student through the learning graph toward a solution. Hints are then provided to the student as applicable, with hint frequency throttled according to a student's programming behavior to support pedagogical efficacy; this is a strategy inspired most directly by previous work by Piech et al. (Piech et al., 2015). As examples, a stopper might receive a higher frequency of just-in-time hints than a mover, or a teacher might configure the system to provide only on-demand hints, so that students must proactively choose to receive them, in the case of a formally assigned and assessed game.

To generate useful hints, we treat student snapshot data as a latent variable in a Hidden Markov Model (HMM); for example, these latent milestones might represent the start of the project, or the correct placement of an important repeat block. We then cluster to identify K milestones across a large set of available snapshots. We cluster gameplay paths according to the probability of a student moving between one state and another and the probability that a particular snapshot started from a particular milestone. We then use these paths, the student's identified programming behavior, and current gameplay state as inputs for hint generation.

With an aim to identify blocks a student might try next when stuck, we model the students' data for a specific HMM cluster and game as a graph, with nodes representing partial gameplay solutions, and edge weights depending on the number of students who have transitioned between two gameplay solutions and reached a successful solution. We then use Dijkstra's algorithm to find the shortest path from the current snapshot to a correct solution, and output the top three blocks that lead the student along that shortest path.

With this inner loop ITS functionality in place, each student interacts with individualized problem-solving feedback while the teacher monitors class progress and supports CT learning when and where needed rather than assuming sole responsibility for class-wide CT delivery and uptake.

4) SAGE Datastore: The Datastore component functions as the database layer for the whole application, consisting of persistent data collection of users profiles, authentication information, games, quests, assessments, and feedback histories.

SAGE Conceptual Architecture

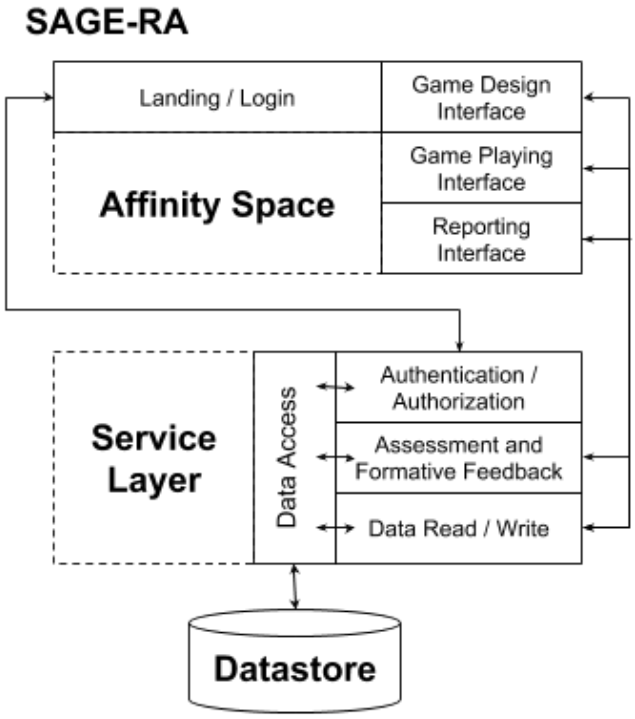

Fig. 2. SAGE-RA: Conceptual Reference Architecture

The application of SAGE-RA relies on its potential for application to a wide array of concrete learning environment software architectures. SAGE-RA achieves this through the following features:

1) Client Independence: While SAGE-RA nicely encompasses several web-based learning environments, it has no explicit networking, client device, or web-specific dependencies. The architecture could be implemented as a standalone application, a scalable distributed learning environment, a classic web application, or in other application models.

2) Assessment Flexibility: While SAGE's concrete architecture includes a novel Formative

Feedback module and employs intelligent hinting (Piech et al., 2015) to implement feedback based on student interactions, there is no prescriptive feedback model included in SAGE-RA. SAGE-RA does specify that feedback should be included and be reactive to assessment; the implementation of providing that feedback is up to the implementer. Grade or score-based feedback, hinting, or simple pass/fail feedback all fit within the confines of the architecture, which may allow for its application to systems strictly in the computer-based testing / assessment subdomain, e.g. certification tests.

3) Game Design Flexibility: SAGE-RA does not specify a set of game paradigms that can be designed in compliant systems. The architecture is as applicable to direct instruction (Helminen, Ihantola, Karavirta, & Malmi, 2012) as constructionist (Weintrop & Wilensky, 2012) game design, and could as easily be applied to games that integrate both approaches (Shabo, 1997), or to future game-based learning methodologies.

Comparison with Code.org® Mapping

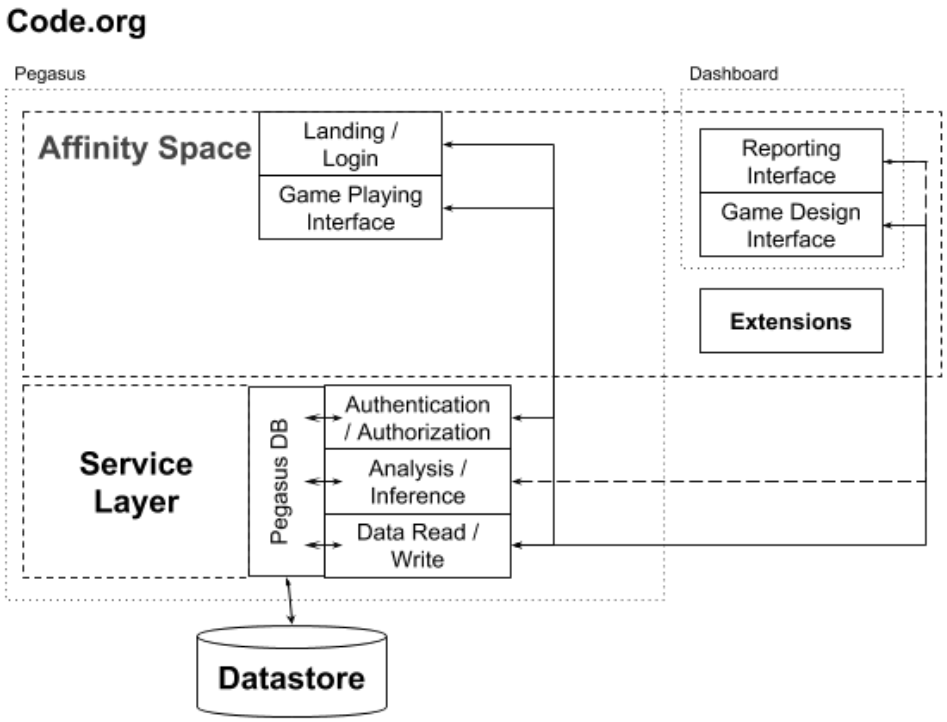

Fig. 3. Mapping SAGE-RA to Code.org®

The primary components of Code.org®'s concrete architecture, as presented in it's GitHub repository(code-dot org, 2019) map to the primary components of SAGE-RA. Subcomponents, such as Code.org®'s Reporting Interface and Game Design Interface, exist within a Dashboard primary component, which in SAGE- RA would be considered a grouping or subcomponent within the Affinity Space primary component. Code.org®'s architecture also supports an Extensions component, which is not included in SAGE-RA. Though Code.org® was developed prior to the creation of a usable reference architecture for gameful learning, it maps well to the general reference architecture presented here. This suggests that such an architecture could be used for future gameful learning environment implementations.

Conclusions

We present the evolution from an existing concrete architecture to a conceptual, and then to a reference architecture, SAGE-RA, and thus extend prior work by Holt and Hassan (Hassan & Holt, 2000) on using conceptual architectures without domain knowledge. We also validate the derived reference architecture against a widely-used web-based gameful learning environment, Code.org®, and classify SAGE-RA as a preliminary, classical, facilitation reference architecture for multiple organizations. We have shown the key components that make up a part of an integrated learning application, that are functionally integrated and yet modular enough to be adapted to a wide variety of desired implementations of a learning system. We hope that this architecture will provide developers of the referenced applications, and future applications in the Gameful Computational Thinking Education and Intelligent Tutoring System domains, an opportunity to validate their designs against a more generalized architecture. This should result in faster roadmap decisions, implementations, and iterations with reduced architectural inefficiencies or errors. We hope this reference architecture will help foster the creation of continuously improving AI-powered learning platforms that encourage effective computational thinking learning that is fun, engaging, and instructive for students and barrier-reducing, supportive, and burden-relieving for teachers.

Discussion and Recommendations

Further mapping analyses of other learning-oriented applications will be useful to further validate the comprehensiveness of SAGE-RA. Bootstraping SAGE for each learning environment, consisting minimally of a school and a grade/class, is an area of focus for future work. In particular, the challenge of data interoperability between SAGE and other systems via a standard data format and exchange protocol, and the question of how to generate mock data for users in a particular unobserved class/level and school, are potential areas for future research. Of particular interest is how SAGE might benefit from data provided by GCTED systems that a given student has used previously, and how SAGE's data might be used by GCTED systems that this student will encounter in later education. Future work for SAGE may include enlisting actual student users to start interacting with the system. Depending on the scope of the learning goals in question, we could create initial task sets and step sets as part of a teaching curriculum or a learning goal, and work with teachers with the appropriate experience and understanding of the curriculum to model the behavior of an effective tutor. By recording the tutor-system-student interactions that follow, we would accumulate a body of interaction data that could serve to inform and train the machine learning models we have built to increase the accuracy, robustness, and the applicability of the learning system.

Acknowledgments

Columbia University's Programming Systems Laboratory is supported in part by NSF CNS-1563555, CCF-1815494 and CNS-1842456.

References

About us — code.org.(n.d.). https://code.org/about. (Accessed: 2018-12-10)

Angelov, S., Grefen, P., & Greefhorst, D. (2009). *A classification of software reference architectures: Analyzing their success and effectiveness.* In *2009 joint working ieee/ifip*

conference on software architecture (wicsa) & 3rd european conference on software architecture (ecsa) (pp. 141–150).

B. J. Ericson, e. a. (2017). *Solving parsons problems versus fixing and writing code.* In *Koli calling conference on computing education research.*

Carvalho, M. (2017). *Serious games for learning: A model and a reference architecture for efficient game development* (Unpublished doctoral dissertation). Eindhoven University of Technology, University of Genoa.

C. Harteveld, e. a. (2014). *A design-focused analysis of games teaching computer science.* Games+Learning+Society.

Cheng, N., & Harrington, B. (2017). *The code mangler: evaluating coding ability without writing any code.* ACM SIGCSE.

code-dot-org. (2019). *Code.org*. https://github.com/code-dot-org/code-dot-org. GitHub.

D. Weintrop, e. a. (2012). *Redefining constructionist video games: marrying constructionism and video game design.* In *Constructionism conference.*

Gee, J. P. (2003). *What video games have to teach us about learning and literacy.* New York: Palgrave.

Hassan, A. E., & Holt, R. C. (2000). *A reference architecture for web servers.* In *Reverse engineering, 2000. proceedings. seventh working conference on* (pp. 150–159).

Helminen, J., Ihantola, P., Karavirta, V., & Malmi, L.(2012). *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* (pp. 119–126). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2361276.2361300 doi: 10.1145/2361276.2361300

Järvelä, S., & Renninger, K. (2014). *Designing for learning: interest, motivation, and engagement.* In *R. Sawyer (Ed.), The cambridge handbook of the learning sciences, second edition.* Cambridge University Press.

J. Helminen, e. a. (2012). *How do students solve parsons programming problems?: an analysis of interaction traces.* International Computing Education Research.

J. M. Randel, e. a. (1992). *The effectiveness of games for educational purposes: a review of recent research.* Simulation and Gaming, 23, 261-276.

Kazimoglu, C., Bacon, L., & Kiernan, M. (2010). *Learning introductory programming through the use of digital games in higher education*.

Kim, T., & Axelrod, S. (2005). *Direct instruction: an educators' guide and a plea for action.* The Behavior Analyst Today, 6, 2.

Lee, M. J., Ko, A. J., & Kwan, I. (2013). *In-game assessments increase novice programmers' engagement and level completion speed.* In *Proceedings of the ninth annual international acm conference on international computing education research* (pp. 153–160).

Leyzberg, D., & Moretti, C. (2017). *Teaching cs to cs teachers: Addressing the need for advanced content in k-12 professional development.* In *Proceedings of the 2017 acm sigcse technical symposium on computer science education* (pp. 369–374). New York, NY, USA: ACM. Retrieved fromhttp://doi.acm.org/10.1145/3017680.3017798doi:10.1145/3017680.3017798

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E.(2010, November). *The scratch programming language and environment.* Trans. Comput. Educ.,

10(4), 16:1–16:15. Retrieved from http://doi.acm.org/10.1145/1868358.1868363 doi: 10.1145/1868358.1868363

Mislevy, R.J., & Haertel, G.D. (2006). *Implications of evidence-centered design for educational testing.* Educational Measurement: Issues and Practice, 25(4), 6-20. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1745-3992.2006.00075.x doi:10.1111/j.1745-3992.2006.00075.x

Owen, V. E., Shapiro, R. B., & Halverson, R. (2013). *Gameplay as assessment: Analyzing event-stream player data and learning using gba (a game-based assessment model)*. In Cscl (1) (pp. 360–367).

Parsons, D., & Haden, P. (2006). *Parson's programming puzzles: a fun and effective learning tool for first programming courses.* In *Australian conference on computing education.*

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R.(1986). *Conditions of learning in novice programmers.* Journal of Educational Computing Research,2(1), 37-55.Retrieved from https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL doi: 10.2190/GUJT-JCBJ-Q6QU-Q9PL

Piech, C., Sahami, M., Huang, J., & Guibas, L. (2015). *Autonomously generating hints by inferring problem solving policies.* In *Proceedings of the second (2015) acm conference on learning @ scale* (pp. 195–204). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2724660.2724668 doi: 10.1145/2724660.2724668

Reiser, B. J., & Tabak, I. (2014). *Scaffolding.* In *R. Sawyer (Ed.),The cambridge handbook of the learning sciences, second edition.* Cambridge University Press.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., …, Kafai,Y. (2009, November). Scratch: *Programming for all.* Commun. ACM, 52 (11), 60–67. Retrieved from http://doi.acm.org/10.1145/1592761.1592779 doi: 10.1145/1592761.1592779

Rupp, A. A., Gushta, M., Mislevy, R. J., & Shaffer, D. W. (2010, Jan.). *Evidence-centered design of epistemic games:Measurement principles for complex learning environments.* The Journal of Technology, Learning and Assessment, 8 (4). Retrieved from https://ejournals.bc.edu/index.php/jtla/article/view/1623

Sadiman, A.S. (2004). *Challenges in education in southeast asia.* Retrieved from http://www.seameo.org/VL/library/dlwelcome/publications/paper/india04.htm

Shabo, A. (1997). *Integrating constructionism and instructionism in educational hypermedia programs.* Journal of Educational Computing Research, 17 (3), 231–247.

Squire, K. (2004). *Replaying history: learning world history through playing civilization iii.* Indiana University: Unpublished Ph.D Dissertation.

United States Census Bureau. (2016, August). *Occupations in information technology.* https://www.census.gov/library/publications/2016/acs/acs-35.html.

Vanlehn, K. (2006, August). *The behavior of tutoring systems.* Int. J. Artif. Intell. Ed., 16 (3), 227–265. Retrieved from http://dl.acm.org/citation.cfm?id=1435351.1435353

Weintrop, D., & Wilensky, U. (2012). *Redefining constructionist video games: Marrying constructionism and video game design.*

Wing, J. M. (2006, March). *Computational Thinking.* Commun. ACM, 49 (3), 33–35. Retrieved from http://doi.acm.org/10.1145/1118178.1118215 doi: 10.1145/1118178.1118215