SAGE Intelligent Hinting, DevOps, and Publication Proposal

Final Report

COMS6901 – Projects in Computer Science, Summer 2018

Alex Dziena / ad3363

Table of Contents

Introduction

Over the course of the Summer 2018 semester, I continued the work done on Intelligent Hinting, DevOps, and made progress on the Publication of SAGE's feasibility study in previous semesters, in particular building on the progress made in Spring 2018. Initially, Tomer Zwi (tz2247) and I proposed an integration of the existing hinting system, the addition of a natural language (chatbot) hinting interface, DevOps enhancements to improve the stability of the platform for continued field studies in the Fall semester, reducing the time-to-productivity for future researchers, and contributing a new reference architecture to the publication draft. These would have addressed the following Epics and Features:

Epic	Feature
Gameful Intelligent Tutoring	Intelligent Hinting MVP
SAGE Integration	DevOps MVP
	Workstream Integration
Survey, Field Study Design, and Publication Strategy	SAGE Feasibility Study & Publication

We ran into significant challenges over the course of the semester, mostly related to merging the finished code from Spring 2018, and so made less progress on the Intelligent Hinting MVP and DevOps MVP than originally proposed.

Gameful Intelligent Tutoring

We proposed to fully integrate the existing intelligent hinting system into the SAGE UI in such a way that future intelligent hinting modules can be easily developed and integrated, and to develop a natural language interface for students to "request" hints by asking questions in a chat interface.

We did not complete UI integration for intelligent hinting modules to be created this semester, or create an NLP-based hint request system. We did work on testing the intelligent hinting module and extending the API for intelligent hinting.

SAGE Integration

We had planned to implement automated configuration management, Newman integration for automated local API testing, improving test coverage of all SAGE codebases, and developing a set of coding standards for the project. We also planned to fully document SAGE's modified Agile project management methodology, and develop a set of Use Cases and Personas for SAGE to improve the onboarding / orientation of future researchers.

We implemented automated configuration management for shared SAGE environments (dev and uat), selecting Puppet as our configuration management system. We further built Puppet modules that allowed for automated setup of researchers' local development environments.

We deferred Newman integration to the Fall 2018 semester, to be combined with automated Selenium web testing.

We identified coding standards for SAGE's Javascript and Python code bases, and published these to the SAGE wiki, but did not include linting or standards checking as automated parts of the build system. However, we deprioritized the documentation of our modified Agile process and Use Cases and Personas because of the low number of contributors this semester. Instead, we prioritized documentation of local environment setup.

Survey, Field Study Design, and Publication Strategy

We focused on improvements to the reference architecture proposed in the original feasibility paper study, created a reference architecture diagram, and updated the existing data visualizations in the paper.

Related Work

DevOps work on SAGE within the SAGE Integration epic this semester focused on improving the time to productivity for researchers in the fall. Specifically, taking inspiration from Lwakatare et al.'s work on the dimensions of DevOps, we focused on identifying a configuration management framework, Puppet, for use in SAGE and developing initial modules for set up of researchers' local development environments[1]. We also drew inspiration from Maruping et al.'s work on code standards in selecting our Javascript and Python standards.

Work on Gameful Intelligent Tutoring was guided by existing work on on-demand vs proactive hinting, Peich's work on automated hint generation, and CIRCSIM-Tutor's implementation of a dialogue-based intelligent tutoring system. Since this semester's focus was on integration of previous semester's work, we leveraged related work to develop tests and extend the API definition for intelligent hinting

We used the framework for deriving reference architectures from existing implementations first presented by Hassan and Holt, and Angelov's et al.'s work on a framework for classifying reference architectures to create our reference architecture diagram. The diagram represents a combination of Angelov's type 3, *classical, facilitation*, and type 5.1, *preliminary, facilitating* reference architecture.

SAGE Integration: DevOps

[Dimensions of DevOps]

Lwakatare, et al. identified four primary dimensions of DevOps practice: collaboration, automation, measurement, and monitoring. Their methodology was a survey of DevOps practitioners. They use their survey results to specify a framework for characterizing DevOps practices[1].

The paper highlights manual configuration management, and the resulting reduction in productivity for developers, as one of the primary problems that DevOps addresses. This inspired us to prioritize selection of a configuration management framework, Puppet, for use in SAGE, and develop the modules needed for automated setup and management of researchers' local development environments.

[Role of collective ownership and coding standards in coordinating expertise in software project teams]

Maruping et al. performed a field study of 509 software developers within 56 project teams. They highlight coding standards as a key factor in software project technical quality, finding that a sense of collective ownership and the existence of coding standards are correlated with higher quality software. The authors exclusively study software developers working within the Extreme Programming(EP) methodology, but the authors found no evidence suggesting that their results would not be generalizable to other development methodologies. In particular, there is nothing in the paper that suggests that collective ownership and coding standards would be less effective within the context of the modified Agile approach used for SAGE[2].

This work inspired our selection of the semistandard set of coding standards for SAGE's Javascript code, and the PEP 8 style guide for SAGE's Python code. The paper highlights a shared understanding of coding standards and ease of understanding of those standards as key factors in their success in enhancing software quality. To help facilitate shared understanding and ease of understanding, we chose to adopt widely-used coding standards instead of creating a custom set of SAGE-specific standards. This should improve both the readability and comprehensibility of the code for new researchers (who may have prior experience with these widely used standards), and the ease of adoption and integration with automated linting and style checking tools.

[Agile Approaches on Large Projects in Large Organizations]
This research explores the apparent conflicts between agile and traditional project management in large organizations first implementing agile practices. It also defines several of the roles and practices used in agile project management, highlighting the importance of communication and stakeholder management[3].

This research has informed our development of a modified agile project management methodology. We deprioritized the documentation of our modified Agile process this semester, but plan to include that documentation in Future Work.

Gameful Intelligent Tutoring

[Hints: Is It Better to Give or Wait to Be Asked?]
Razzaq, et al. found that students learned more reliably when they were provided with a mechanism to receive hints-on-demand rather than being provided with hints proactively in a "just-in-time" hinting system[4]. Because this effect was more pronounced in students who asked for a larger number of hints, the study may suggest that the relative benefit of hints-on-demand vs proactive hints increases as a student needs increasing amounts of assistance. This paper provided us the inspiration and rationale for a natural language interface to an on-demand hinting system. However, given the additional work needed to merge the contributions from Spring 2018 and get that code working with our continuous integration system, and the need to set up the other Summer 2018 students with usable development environments and orient them to the code base, the natural language hinting interface had to be deprioritized.

[Autonomously Generating Hints by Inferring Problem Solving Policies]
Piech, et al. performed analysis on a large amount of student solutions data gathered from Code.org and proposed several different path modeling algorithms used to learn a Problem Solving Policy (PSP), a policy that returns a suggested next partial solution for all partial solutions to a given puzzle, i.e. a PSP π is defined as $s' = \pi(s)$for all $s \in S$, with Sbeing the set of all possible solutions. The algorithms used to learn this PSP were compared, with a class of algorithms called Desirable Path algorithms (Possion Path and Independent Probable Path) performing the best - producing PSPs closest to the paths contained in an expert-labeled set of suggested paths[5].
“Poisson Path”-based PSP learning and hint generation were implemented last semester[6]. This semester, we added additional testing for hint generation and extended the API for retrieving hints.

[Delivering Hints in a Dialogue-Based Intelligent Tutoring System]
Zhou, et al present an implementation of CIRCSIM-Tutor, a dialogue-based intelligent tutoring system. The system used a labeled corpus of dialogue (questions and answers) from expert tutors to develop a set of hinting strategies used to select appropriate hint templates and parameters as responses to classes of questions. CIRCSIM-Tutor does not support online-updating, but is context aware of it’s previous interactions with a student during a single session[7].
As stated above in “**Hints: Is It Better to Give or Wait to Be Asked?**,” we did not implement a natural language based chatbot to provide intelligent hints this semester. Instead, we prioritized orientation of new researchers, and work to integrate Spring 2018’s work into the existing SAGE codebase.

Publication

[The concept of reference architectures]
Cloutier, et al. examine reference architectures with the goal of providing a more precise definition of their components and purpose[8]. They propose that the value provided by reference architectures lies in the distillation of (in many cases) thousands of person-years of work, a shared baseline for multiple, often cross-functional teams, and guidance for future work. We used this paper’s architecture evaluation methodologies and structure / component suggestions to guide the development of a structure and definition of components in the creation of a reference architecture diagram for SAGE.

[A framework for analysis and design of software reference architectures]
This paper provides a tool in the form of an analysis and design framework, for the creation of software reference architectures based on three primary dimensions: context, goals, and design. The paper goes on to define five types of reference architectures into which architectures under analysis can be classified:

1. classical, standardization architectures for use in multiple organizations

2. classical, standardization architectures for use in a single organization
3. classical, facilitation architectures for use in multiple organizations
4. classical, facilitation architectures for use in a single organization
5. preliminary, facilitation architectures for use in multiple organizations

The paper validates the framework by applying it to analysis of 24 reference architectures[9]. Our reference architecture diagram represents a combination of Angelov's type 3, *classical, facilitation*, and type 5.1, *preliminary, facilitating* reference architectures. A type 3 architecture is designed for multiple organizations by an independent organization, and we've developed SAGE's reference architecture diagram to be useful to any practitioners working on implementing an interactive system designed to teach computational thinking using structured, game-based learning modules integrated with hinting and instructor feedback mechanisms. A type 5.1 architecture describes a preliminary architecture that facilitates the design of architectures of future systems. As SAGE is not yet fully available, our architecture may be useful as a basis for the design of other hint-based / game-based learning systems, even outside of the realm of computational thinking.

[The visual display of quantitative information]
This book is widely used in industry and academia, and provides an exploration and set of recommendations for the design of statistical graphics, charts, tables[10]. It also provides an analysis framework for selecting appropriate data visualizations for a given data set, attempting to optimize for precision, efficacy, and speed of analysis. We used Tufte's work extensively when revising the data visualizations in the field study paper.

[A Reference Architecture for Web Servers]
Hassan and Holt present a framework for reverse engineering a reference architecture for a domain from a concrete implementation, and apply this framework to the domain of web servers. They go on to validate this architecture against several leading implementations of the time, including Apache which is still perhaps the most widely used web server[11]. Grosskurth and Godfrey further validate this approach by applying the same framework to the domain of web browsers, five years after the publication of Hassan and Holt's paper[12].

We used Hassan and Holt's approach to create SAGE-RA, and derived the reference architecture diagram from SAGE's concrete implementation. In Fall 2018, we will use this same approach to validate the derived reference architecture against code.org, scratch.mit.edu, and Blockly Games.

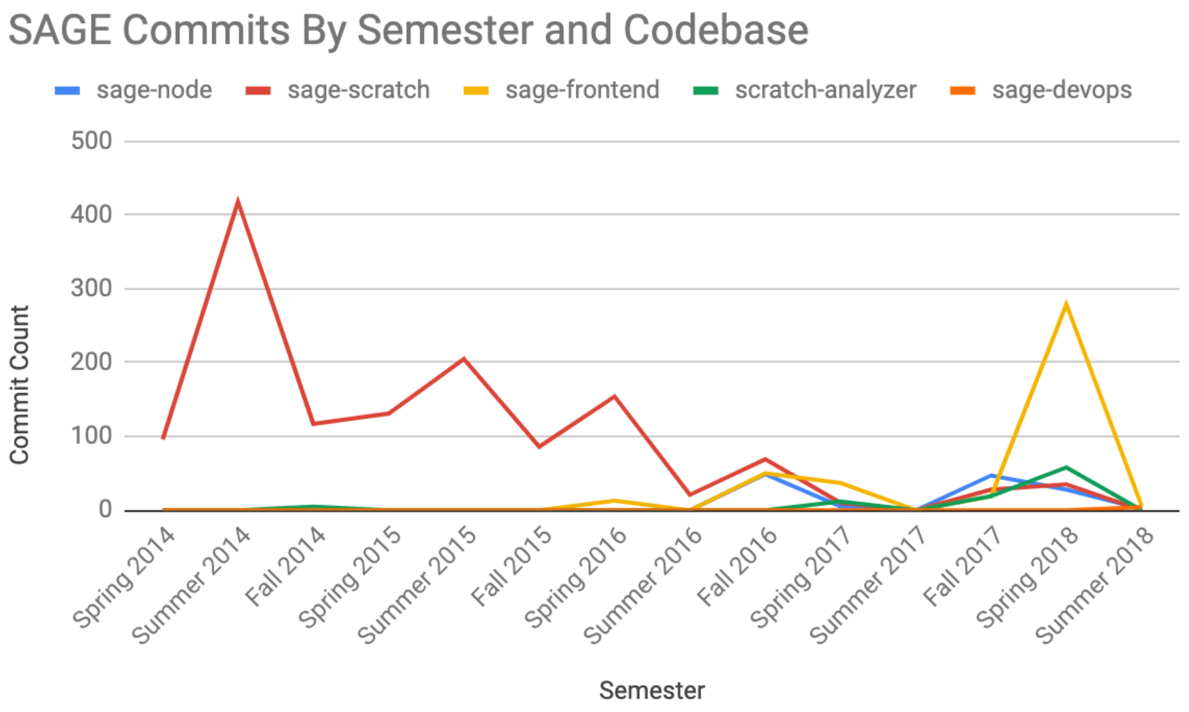
Accomplishments

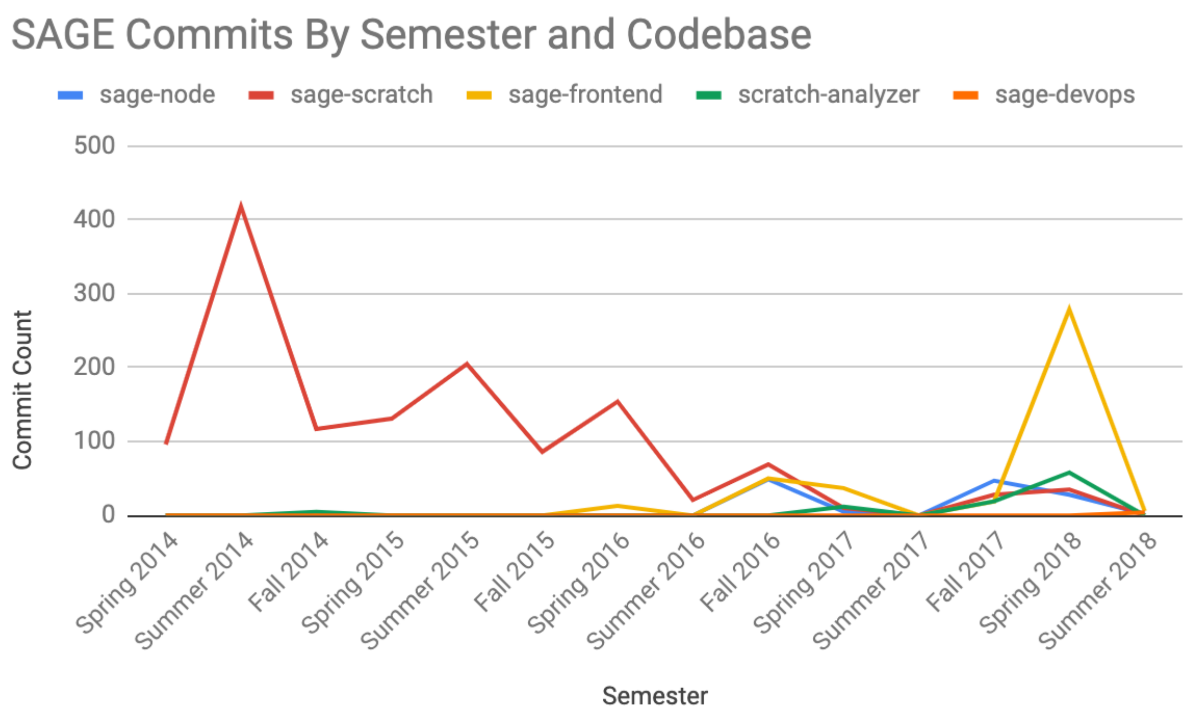
This semester focused on integration of past semesters' work into the SAGE codebase, optimizing for the productivity of researchers in Fall 2018, and developing a useful reference architecture for publication. We greatly improved the stability of the codebase by integrating the work of multiple contributors across several SAGE codebases. We improved the code quality for future semesters by introducing code standards for Javascript and Python, selecting and

implementing a configuration management system for set up and management of local environments for new researchers. We introduced testing for the existing intelligent hinting system, and improved the API for retrieving intelligent hints.
We also developed a reference architecture diagram, which is useful as a reference architecture for online Gameful Learning environments, and can be validated against multiple block-based novice programming instruction systems which are in wide-use today.

SAGE Integration: DevOps

SAGE's volume of commits per semester, and the language-diversity of its codebase increased significantly in Spring 2018. Given this increase, standardization and code health, in the form of coding standards, and the stability of the codebase, in the form of clean integrations with work from previous semesters, has become critical to the success of the project.





Code Standards and Linting

1. ### Standards and Toolsets

   We selected the following standards and toolsets:

Language	**Standard**	**Toolset**
Javascript / Node.js	standard with semicolons[13]	semistandard
Python	PEP 8 style guide[14]	pylint

   Usage of common, well maintained toolsets may allow new researchers to use tools and standards they are familiar with, may help prepare future researchers for work on other projects that also use these standards, and provides us the ability to automatically fix many style and linting errors after each commit.

2. ### Configuration management

   We performed an in-depth analysis of three configuration management systems to determine their suitability for integration into SAGE . We developed a rubric which took into account seven suitability factors:

1. **Maturity:** The number of years the system has been generally available. This was considered a proxy for the stability and ease-of-adoption for the system
2. **Complexity:** This was a qualitative judgement that I and Tomer applied to each system, based on the number of configuration options available and the comprehensibility of the configuration options to a novice practitioner.
3. **Fault tolerance / Consistency, Availability, and Partition tolerance (CAP theorem):** This judged each system based on three criteria:
   i. *Consistency:* How likely is it that every application of the configuration is applying the most recent, canonical representation of that configuration?
   ii. *Availability:* Every attempt to apply a configuration is successful, without guaranteeing that the most recent, canonical representation of the configuration is being applied?
   iii. *Partition Tolerance:* How resilient is the configuration management system to failures of the source control management system or any central services (e.g. a central Puppet server)?

Our analysis resulted in the selection of Puppet as the most applicable configuration management system for SAGE.

System	Score	Maturity (# of years of general availability)	Complexity (1-10)	Fault tolerance / Consistency, Availability, and Partition tolerance (CAP theorem)(1-10)	Estimated time to implement in weeks	Maintenance overhead (1-10)	Interface simplicity (1-10)	Use-case coverage (1-100)
Chef	201.7777778	9	7	3	8	7	5	8
Puppet	404.3076923	13	6	3	4	4	7	8
Ansible	202	6	2	6	8	5	5	6

After selecting Puppet, we implemented modules for the initial set up and continuous management of researchers' local development environments in the sage-devops repository.

Gameful Intelligent Tutoring

Intelligent Hint Integration

We merged multiple broken merges for the game controller changes developed in Spring 2018. We developed several tests for the sage-node and sage-frontend repositories that exercised login for intelligent hinting, and extended the API for retrieving hints through sage-frontend by codifying the user-type parameter. Merging of Spring 2018 code in order to increase other researchers' productivity throughout the Summer 2018 semester was the primary goal, so a large amount of time was spent fixing broken merges and integrating this code.

Publication

(note: derived from Fall 2018 based on the initial RA diagram developed in Summer 2018)

Reference Architecture

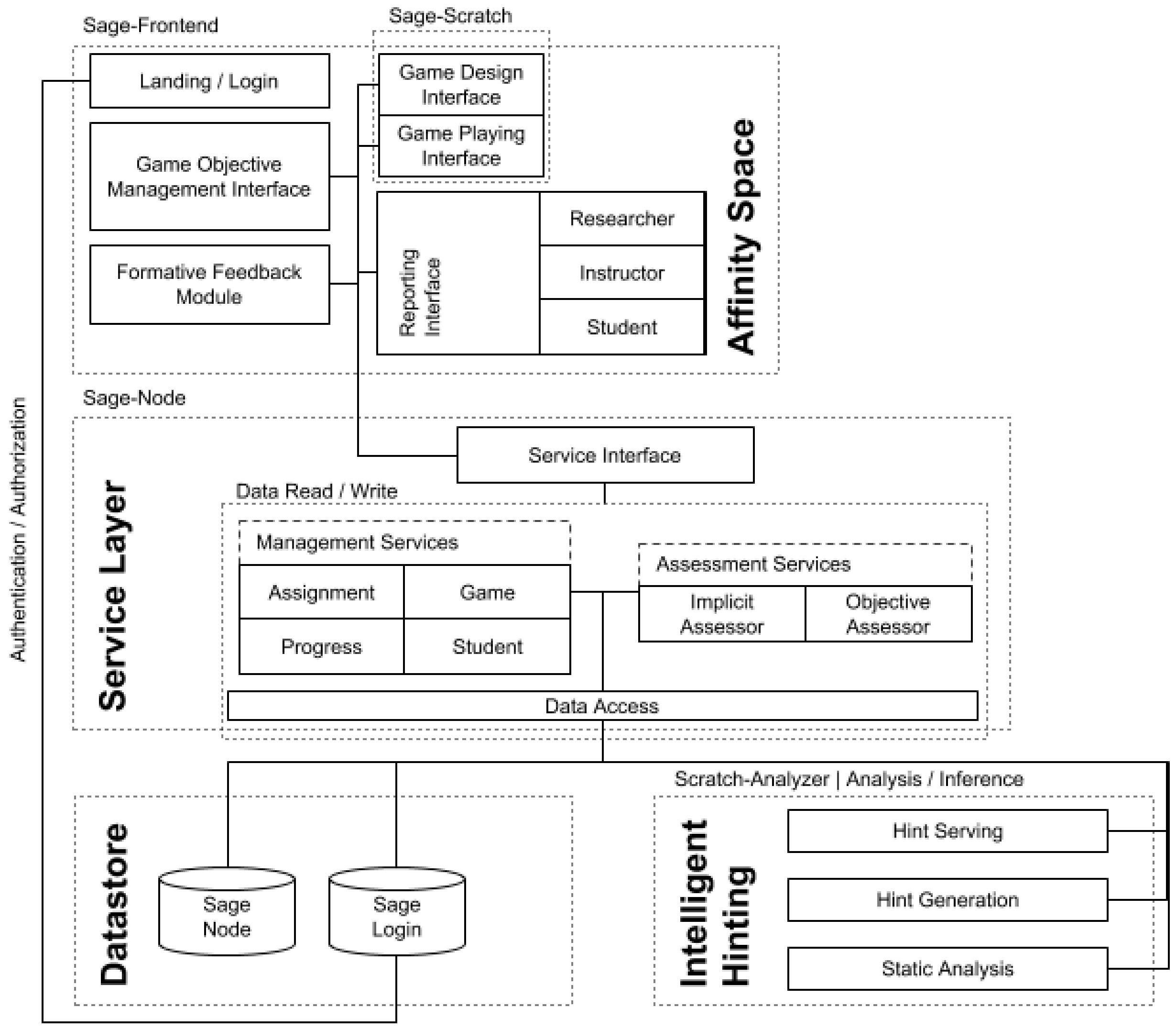
While discussing the feasibility study for SAGE, it became apparent that a reference architecture could be of use to other researchers and implementers of gameful learning systems. The architecture we created is useful in the design and implementation of a learning environment with a consolidated user interface for constructionist and instructionist learning.

We developed a reference architecture diagram from our existing implementation and documentation (SAGE). We will discuss our observations of architectures within the domain of gameful instruction, and present a roadmap for future work the community can leverage to influence and build intelligent systems that accelerate the proliferation of effective and efficient dissemination of computational thinking throughout middle school and beyond.

We developed an initial concrete architecture based on our experience with the project and analysis of the existing codebases. Assessment and Formative Feedback within the Intelligent Hinting layer were identified as two of SAGE’s most unique and defining features.

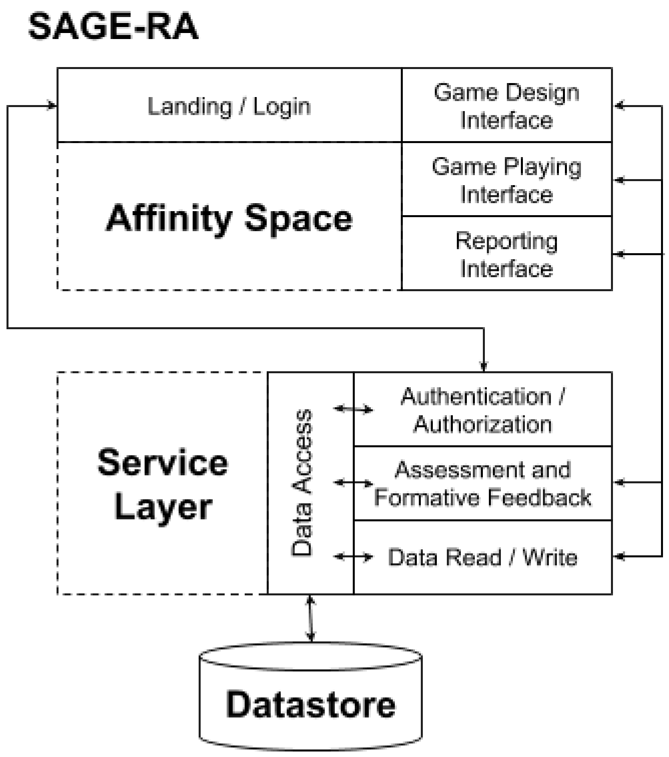
Based on Hassan and Holt's work on reverse engineering of reference architectures [11], we used this concrete architecture to derive a conceptual architecture, based on a generalization of the concrete architecture and plans for future development. From the conceptual architecture, we created a conceptual reference architecture diagram.

SAGE (Concrete Architecture)



Reference Architecture

While developing our architecture diagram, we focused on generalization; i.e. the applicability of the architecture to gameful learning applications in general. Flexibility was a key factor in wide applicability, and thus we focused on three features of flexibility:



Client Independence

While SAGE-RA nicely encompasses several web-based learning environments, it has no explicit networking, client device, or web-specific dependencies. In particular, this model could be implemented as an offline, standalone model, or just as easily as a distributed native application for mobile devices. Hybrid implementations (e.g. web-based, mobile-native, and standalone) could reuse all of the components, given the authentication and authorization modules, and the datastores, had network access to centralized servers.

Assessment Flexibility

We found that flexibility in a feedback module would be important to developers of gameful learning systems in diverse fields. In particular, dialogic pedagogy (interactive feedback)[15] has been found to be effective in teaching novel concepts, and SAGE does not yet incorporate this. A natural language chatbot, as proposed at the beginning of the semester, might be an effective mechanism for this. Our reference architecture does specify that feedback should be included and be reactive to assessment; however, the implementation of that feedback, and the pedagogical method through which it is implemented, is up to the implementer. The reference architecture accommodates grade or score-based feedback, hinting, simple pass/fail feedback, or any other feedback method.

Game Design Flexibility

We designed our architecture to be applicable to instructionist and constructionist game design. The architecture is flexible enough to be applied to games that integrate these approaches, use only instructionist or constructionist design, or use as yet undefined methods.

Future Work: Next Steps

Given the focus on code integration, improvement of researchers' time to productivity, and creating a reference architecture diagram, there are clear next steps for future research:

1. **DevOps**: We extended the intelligent hinting API this semester which surfaced the opportunity for increased testing of APIs throughout SAGE. Also, there would be significant benefits in expanding and improving coding standards throughout the codebase, and improving code quality through increased test coverage. Concrete next steps could include:
    a. Integration testing using Newman for APIs, and Selenium for web UI testing
    b. Documentation of SAGE's Code Standards for each repository and language
    c. Post commit hooks to prevent build-breaking changes from integrating into the main ("development") branch
    d. Improving test coverage to 60% per repository
2. **Gameful Intelligent Tutoring:** We had initially proposed integrating intelligent hinting through a chatbot UI, and based on Zhou et al.[7] and Razzaq et al.[4], we feel that this integration would increase the useful of the existing intelligent hinting inference system:
    a. Implementing a "chatbot" interface to SAGE's Gameful Intelligent Tutoring system
    b. Completing the integration of intelligent hinting with the SAGE UI
3. **Publication** - Creating a draft for the publication of a reference architecture for SAGE, and validating that architecture against other gameful instruction implementations would be a useful next step for publication.

Conclusion

Our work this semester was primarily focused on integration of code from previous semesters, creating a configuration management solution to improve the time-to-productivity for future researchers, and developing an initial reference architecture for SAGE. We initially proposed the addition of a natural language (chatbot) hinting interface, implementation of Newman testing, documenting our modified Agile project management approach and SAGE Use Cases and Personas, and integration of the existing intelligent hinting system into the SAGE UI, but reprioritized these in favor of integrating the Spring 2018 work and ensuring that our peers this semester could be productive.

We created a useful concrete reference architecture, and initial version of a conceptual reference architecture, for publication. We improved the stability of the codebase by integrating the work of multiple contributors across several SAGE codebases, and integrated code from Spring 2018 and previous semesters following the migration to github. We developed a rubric for selecting a configuration management system for SAGE and implemented the module necessary for automatic configuration, deployment, and ongoing management of researchers'

local development environments. We identified code standards for Javascript and Python, improved testing for the existing intelligent hinting system, and improved the API for retrieving intelligent hints.
We identified multiple opportunities for future work, including validating our reference architecture against code.org, scratch.mit.edu, and Blockly Games.

References

[1] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of DevOps," in *Agile Processes in Software Engineering and Extreme Programming*, 2015, pp. 212–217.

[2] L. M. Maruping, X. Zhang, and V. Venkatesh, "Role of collective ownership and coding standards in coordinating expertise in software project teams," *European Journal of Information Systems*, vol. 18, no. 4, pp. 355–371, Aug. 2009.

[3] B. Hobbs and Y. Petit, *Agile Approaches on Large Projects in Large Organizations*. Project Management Institute, 2017.

[4] L. Razzaq and N. T. Heffernan, "Hints: is it better to give or wait to be asked?," in *International conference on intelligent tutoring systems*, 2010, pp. 349–358.

[5] C. Piech, M. Sahami, J. Huang, and L. Guibas, "Autonomously Generating Hints by Inferring Problem Solving Policies," in *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, Vancouver, BC, Canada, 2015, pp. 195–204.

[6] Z. Y. D. Luo, "Intelligent Hinting 1.1 Final Report," SAGE, , 2018.

[7] Y. Zhou, R. Freedman, M. Glass, J. A. Michael, A. A. Rovick, and M. W. Evens, "Delivering Hints in a Dialogue-Based Intelligent Tutoring System," in *AAAI/IAAI*, 1999, pp. 128–134.

[8] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone, "The concept of reference architectures," *Syst. Eng. Electr.*, 2009, doi: 10.1002/sys.20129. [Online]. Available: http://doi.wiley.com/10.1002/sys.20129

[9] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Information and Software Technology*, vol. 54, no. 4, pp. 417–431, Apr. 2012.

[10] E. R. Tufte, "THE VISUAL DISPLAY OF QUANTITATIVE INFORMATION," *J. Healthc. Qual.*, vol. 7, no. 3, p. 15, Jul. 1985.

[11] A. E. Hassan and R. C. Holt, "A reference architecture for Web servers," in *Proceedings Seventh Working Conference on Reverse Engineering*, 2000, pp. 150–159.

[12] A. Grosskurth and M. W. Godfrey, "A reference architecture for Web browsers," *21st IEEE International Conference on Software Maintenance (ICSM'05)*. 2005 [Online]. Available: http://dx.doi.org/10.1109/icsm.2005.13

[13] *standard*. Github [Online]. Available: https://github.com/standard/standard. [Accessed: 21-Apr-2021]

[14] G. Van Rossum, B. Warsaw, and N. Coghlan, "PEP 8: style guide for Python code," *Python. org*, vol. 1565, 2001 [Online]. Available: http://greentv.mobiloitte.com/FTP-LOCATION/CoodingGuidence/images/Style%20Guide%20for%20Python%20Code%20_%20Python.org.pdf

[15] F. Hardman and J. Abd-Kadir, "Classroom discourse: towards a dialogic pedagogy," *The international handbook of English, language and literacy*, pp. 254–264, 2010.