SAGE DevOps, Gameful Intelligent Tutoring, and Publication

Final Report

COMS6901 – Projects in Computer Science, Fall 2018

Alex Dziena / ad3363

Lily Yu Li / yl4019

Table of Contents

Introduction

Over the course of this semester we've extended the DevOps work done in previous semesters, prepared a preliminary draft of a reference architecture for a computational thinking-focused educational platform based on SAGE's current implementation, and integrated the existing Gameful Intelligent Hinting codebase with the rest of SAGE.

Specifically, we implemented code-style checking and documentation[1], protection of mainline code branches through gated pull requests[2], performed a clean merge of Summer 2018 and

Spring 2018's conflicting Parson's Puzzles work, and added code coverage[3] checking of all primary SAGE codebases. We developed a reference, conceptual, and concrete architecture for SAGE, and validated the reference architecture against code.org's implementation[4]. We have created an API, frontend interface, and bridge to our inference and analyses codebases to integrate Gameful Intelligent Hinting work done in this and past semesters with the SAGE UI.

We addressed the following Epics and Features:

Epic	Feature
SAGE Integration	DevOps MVP
	Workstream Integration
Gameful Intelligent Tutoring	Intelligent Hinting 1.1
Survey, Field Study Design, and Publication Strategy	SAGE Feasibility Study & Publication

Related Work

DevOps work on SAGE to date has been primarily focused on automation and improving the time-to-productivity for new researchers. Continuous integration and deployment, and configuration management for shared and local development environments, has allowed us to shift focus to overall project health, and avoiding regressions. Related work in DevOps and code health include Professor Kaiser et al.'s work on parameterized random testing, Maruping et al.'s work on code standards, and Lwakatare et al.'s work on the dimensions of DevOps.

Our 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.

Our initial draft of SAGE-RA: A Reference Architecture for Gameful Learning was based on a framework for deriving reference architectures from existing implementations first presented by Hassan and Holt, and guided by Angelov's et al.'s work on a framework for classifying reference architectures. Specifically, SAGE-RA is 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. performed a survey of DevOps practices and identified four primary dimensions of DevOps practice: collaboration, automation, measurement, and monitoring. The paper goes on to specify a conceptual framework for characterizing DevOps practices[3].

This paper, in particular the sections on collaboration and automation, helped guide the prioritization and implementation of DevOps work items this semester including:

1. Pull requests and code reviews

2. Integration of the Parson’s Puzzle Spring 2018 and Summer 2018 work

[Role of collective ownership and coding standards in coordinating expertise in software project teams]
This paper explores expertise coordination as an important emergent process through which software project teams manage software development challenges, in particular within the framework of Extreme Programming (XP). Maruping et al examine the relationship between collective ownership and coding standards with software project technical quality in a field study of 56 software project teams comprising 509 software developers, and found that collective ownership and coding standards play a role in improving software project technical quality. They find that coding standards strengthens the relationship, resulting in higher technical quality[5].

This work inspired the SAGE coding standards as documented in the SAGE wiki[1] and guided our implementation of automated linting and code-standards checking within our continuous integration framework.

[Parameterizing random test data according to equivalence classes]
Professor Kaiser, et al. present a framework for parameterized random test case generation for machine learning applications. As stated in this paper, there is no reliable test oracle for ML applications; i.e. we can not, with reasonable confidence, predict the correct output for a given random input[6].

This research guided the implementation of a “fuzz” testing data generation framework, that generates inputs to the behavior detection engine in SAGE’s Gameful Intelligent Tutoring codebase, which will be used after end-of-semester code commits to generate a set of ML test cases.

[Gitflow]
Per Phillips’ survey on branching and merging practices[7], Git was used by the second highest number of survey participants (narrowly less respondents than were using Subversion), and some of the primary determinants of success, or satisfaction with, a branching strategy were 1) minimizing merge conflicts, 2) increasing the frequency of upstream merges, and 3) using Experiment, Feature, and Release branches. GitFlow[8] is a branching and merging workflow, using Git, that minimizes merge conflicts, enables Continuous Integration systems to perform frequent upstream merges, and accommodates experiment, feature, and release branches.

SAGE’s pull-request based branching and merging policy is an implementation of GitFlow. It has the added advantage of automated and enforced code review and build validation for all commits.

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[9]. 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.

[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 S being 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[10].

"Poisson Path"-based PSP learning and hint generation were implemented last semester[11]. This semester, we integrated the hint generation engine with SAGE's UI.

[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 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[12].

We used a strategy similar to CIRCSIM-Tutor to develop a Tensorflow-based chatbot prototype built on Trigram analysis of a "dummy" question and answer corpus.

The CIRCSIM-Tutor paper also provides an evaluation framework for the efficacy of different hinting strategies, and a set of heuristics to select appropriate hint types. This methodology could be used in future semesters to evaluate the hinting strategies learned by our chatbot, and display results in SAGE's researcher interface[12, pp. 128–134].

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. 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 will reference the architecture evaluation methodologies and structure / component suggestions put forth by this paper while redeveloping SAGE's reference architecture included in the feasibility study draft[13].

[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. We plan to use the framework proposed by this paper to design and evaluate our updated reference architecture[14].

[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. 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 will use this book as a reference when evaluating and potentially redesigning the data visualizations in the feasibility study draft[15].

[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[16]. 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[17].

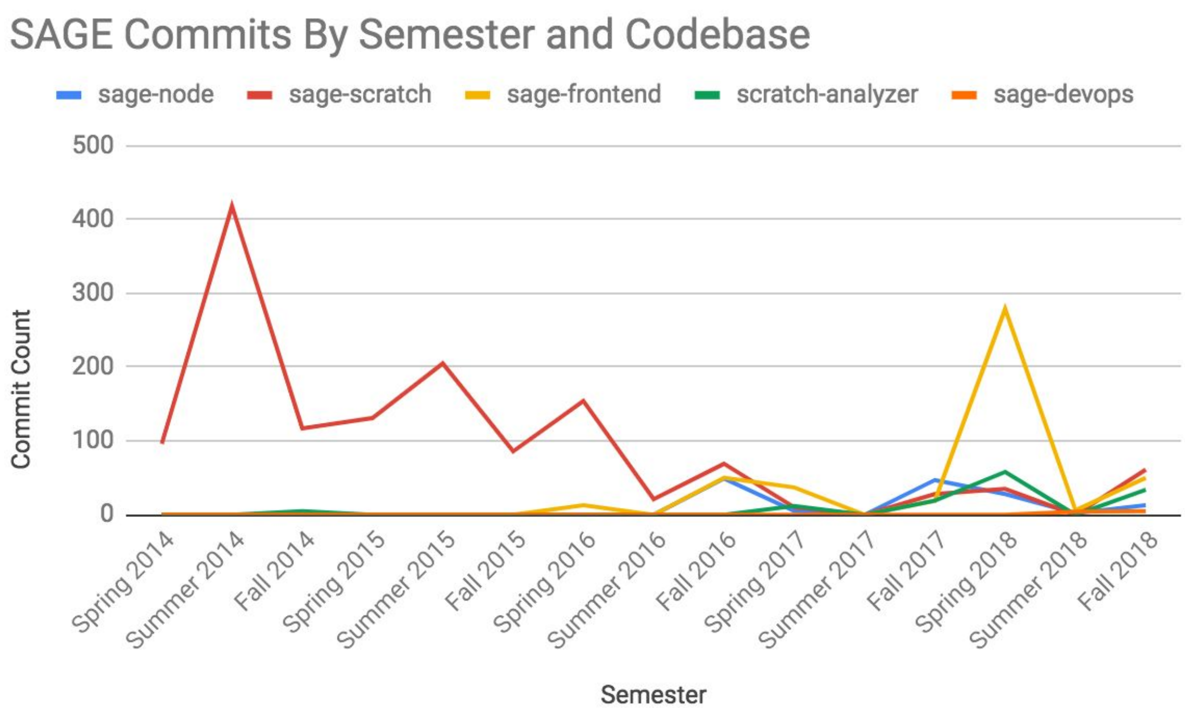
We apply a similar approach in our derivation and validation of SAGE-RA, deriving the reference architecture from SAGE's concrete implementation, and validating the derived reference architecture against code.org, and providing an outline for validation against scratch.mit.edu, and Blockly Games.[4]

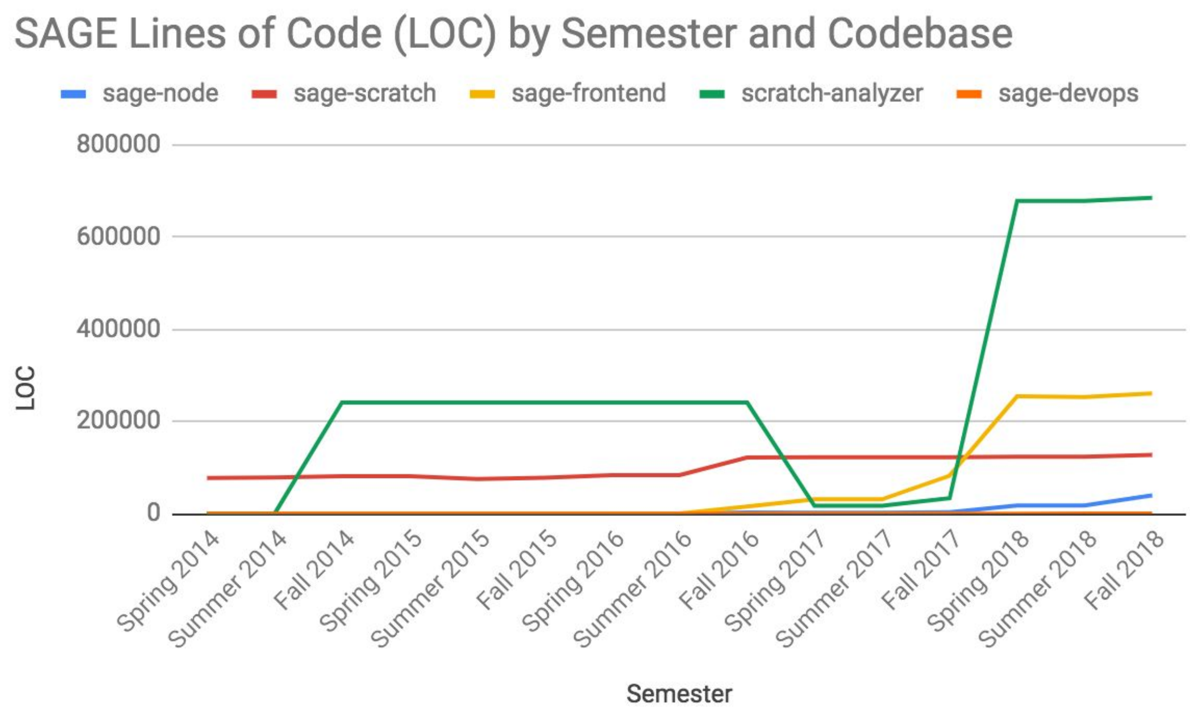
Accomplishments

Over the course of the semester, our team has accomplished several significant milestones for SAGE. We have provided enhanced stability of the codebase through pull requests and test coverage measurement. We have enhanced the quality of the codebase through automated linting, code standards testing, and documentation[1]. We have integrated and extended the existing intelligent hinting and Parson's Puzzle work in a flexible and sustainable way. Also, we have begun work on SAGE-RA: A Reference Architecture for Gameful Learning[4], which provides a concrete architecture for SAGE, a reference architecture for online Gameful Learning environments, and a validation against code.org, perhaps the most widely used block-based novice programming instruction system available today.

SAGE Integration: DevOps

Since the inclusion of other researchers, SAGE's volume of commits per semester, and the diversity of its codebase has increased significantly. Code health, test coverage, and build stability has become critical to researcher productivity as a result.





Code Standards and Linting

1. Documentation

   Documentation has been added to SAGE's wiki[18] detailing code standards for each of SAGE's primary programming languages. Documentation also includes guidelines for introducing new style guides for newly adopted languages, and provides details on integrations with our CI / CD systems.

2. Standards and Toolsets

   We used the following standards and toolsets:

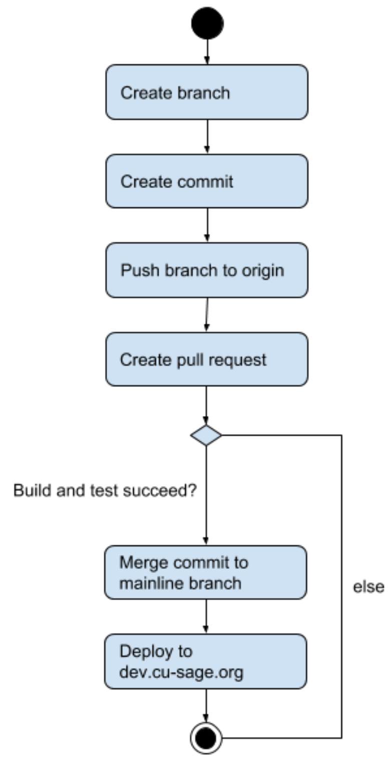
Language	**Standard**	**Toolset**
Actionscript	Apache's FlexSDK Coding Conventions[19]	FlexPMD
Java	Google Java Style Guide[20]	checkstyle
Javascript / Node.js	standard with semicolons[21]	semistandard
Python	PEP 8 style guide[22]	pylint
Ruby / Puppet	Puppet 5.0 Style Guide[23]	puppet-lint

Other Languages	Shellcheck Style Guide[24]	shellcheck

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.

3. Build system integration

Each linting / static analysis tool listed above will be run on every commit to all SAGE repos. At present, semistandard is integrated into the sage-frontend CI/CD system, which will be followed by FlexPMD, checkstyle, pylint, puppet-lint, and shellcheck (by order of priority). After all work is committed at the end of this semester, we will fix all linting and style errors in codebases with a style checking toolset that supports automatic style fixes. This includes our Java and Javascript codebases, and can be extended to the other codebases through the use of extensions and other open source tools.



Gameful Intelligent Tutoring

Intelligent Hint Integration

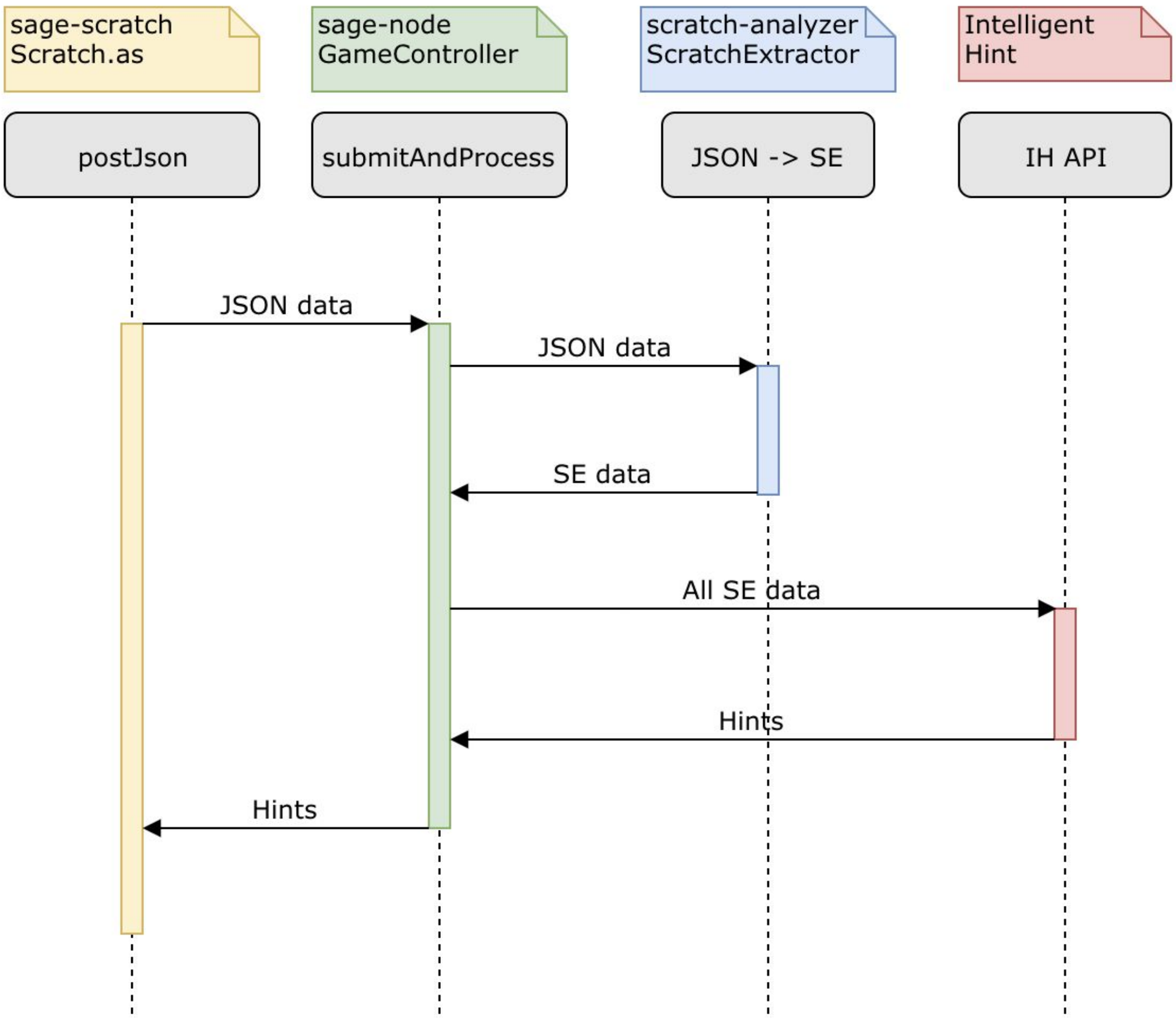
Data Flow

The integration consists of four parts:

1. sage-scratch: the UI component (ActionScript)
2. sage-node: the information center where data is retrieved and stored (NodeJS)
3. scratch-analyzer Java: the data processor that extracts and analyzes the game data
4. scratch-analyzer Intelligent Hinting: the hint generator (Python)

We dispatch requests and serve data through sage-node so that we only need to maintain one interface for data exchange and one schema for each data collection.

Data flow diagram:



Data flow description:

1. Sage-scratch sends the game snapshot as a JSON string to sage-node on a mouse click event.
2. Sage-node uploads the JSON string as a file to the database.
3. Sage-node converts the JSON string to an SE string with block IDs and another SE string without block IDs and uploads both SE files to the database.
4. Sage-node downloads and sends all SE data associated with the current game to Intelligent Hinting.
5. Intelligent Hinting responds with the updated hints.
6. Sage-node forwards those hints to sage-scratch.
7. Sage-scratch processes the new hint information and resets the blocks to suggest and the hint timer.

Implementation

1. **Node Utilities**

We implemented some utilities for the integrated process in sage-node. The new utilities are only used in the game controller for now but can also be applied to other places where such functionalities are required.

- **Run Java code in sage-node**
  Since we need ScratchExtractor to convert .json files to .se files, we want to run java code in sage-node. We implemented this with the node-java module. Previously, ScratchExtractor could only read and write from the local file system. We modified ScratchExtractor to support parsing a single JSON string and return a single SE string. The java project is exported as scratchJava.jar and added to the sage-node branch. The required files for the java project are also added in sage-node/FilesRequired. In the future, whenever someone modified the java code in sage-scratch, the scratchJava.jar should be updated as well so that the Java utilities are able to run the latest Java code. The java utilities are implemented in app/utils/javaUtils.js. Method extractJson takes a JSON string and returns an SE string.

  The following API was created for testing the extractJson method:
  - ➔ /games/jsonToSe/:showId
    - ◆ Parses a JSON string to an SE string.
    - ◆ The SE string will contain block IDs if :showId is true.
- **Store files in the database**
  To connect different code areas and get rid of the dependency on the local file system, we will store files in the database using MongoDB's GridFS. In addition to the file content, we can also store important information such as timestamp, student ID, game ID, and objective ID in the metadata. GridFS also allows us to fetch files with metadata. All utilities for file storage have been added to sage-node/app/utils/fileUtils.js. Right now, we have implemented postDbFile, getDbFile and deleteDbFile to directly upload, download and delete files from the database. Moreover, we have added uploadJson, uploadSe and downloadSeFiles which would return Promises so that the data flow can be created as a sequence.
  1. uploadJson: upload JSON string to the database as a JSON file.
  2. uploadSe: upload SE string to the database as an SE file.
  3. downloadSeFiles: download all SE files with the matching metadata as a JSON array.

  The following APIs were created for testing the file utilities:
  - ➔ /games/uploadJson/:studentID/:gameID/:objectiveID
    - ◆ Uploads a JSON string for a game snapshot.
  - ➔ /games/uploadSe/:studentID/:gameID/:objectiveID/:hasBlockIds
    - ◆ Uploads an SE string for a game snapshot.
    - ◆ The SE string contains block IDs if :hasBlockIds is true.
  - ➔ /games/downloadSe/:studentID/:gameID/:objectiveID/:hasBlockIds
    - ◆ Downloads all SE data for a specific game.
    - ◆ The SE data will contain block IDs if :hasBlockIds is true.
  - ➔ /games/file/:filename
    - ◆ Downloads a file by filename.
  - ➔ /games/file/delete/:filename

◆ Deletes a file by filename.

2. **Hint Requests and Responses**

Sage-node bundles the contents of all .se files associated with a game session and sends them to the intelligent hint system to request hints. The bundled data is a JSON object returned by fileUtils.downloadSeFiles.

Request example:

```
{
  "seFiles": [
    {
      "content": "<<Object Stage>>",
      "timestamp": 1544503546582
    },
    {
      "content": "<<Object Stage>>\n\t\twhenGreenFlag",
      "timestamp": 1544503573786
    }
  ],
  "info": {
    "studentID": "stu123",
    "gameID": "game123",
    "objectiveID": "obj123",
    "hasBlockIds": false
  }
}
```

The “seFiles” field stores a list of SE data order by their timestamps. The content of the SE data is an SE string. The common attributes of these SE data can be found in the “info” field. The SE data sent to Intelligent Hint do not contain block IDs as the intelligent hint algorithms only use block names.

We expect two types of response from Intelligent Hint, both with a “hints” field that contains a list of suggested blocks. The type 1 response has an extra field “nextAutoHintTime”, which stores the timestamp when the hints should be shown to the user automatically. Both types of response will be used to replace the blocks to suggest in the scratch game. If the hints list is empty, then no hints will be available to the user after updating the blocks.

- Response type 1 example:

```
{
 "hints": ["wait:elapsed:from:", "randomFrom:to:"],
 "nextAutoHintTime": 1544503583786
}
```

  Hints will be shown to user at nextAutoHintTime automatically or on demand.

- Response type 2 example:

```
{
 "hints": ["wait:elapsed:from:", "randomFrom:to:"]
}
```

  Hints will be shown to the user on demand only.

3. **Hint Timing**

Previously, the hint timing logic was implemented in sage-scratch and was completely separated from the user behavior and the intelligent hint code area, so we were not able to take advantage of the intelligent system to generate smart hint timing. With the new integrated process, we handed the control of automatic hint timing to the intelligent hint system by taking a “nextAutoHintTime” from the hint response and using it to set the hint timer. Also, the previous logic posted the game snapshot every second. But frequent updates will introduce a lot of redundant requests, especially since it is unlikely that the game snapshot changes every second. Therefore, we modified the logic to only post JSON data after a mouse click, when there are potential changes in the snapshot.

When sage-scratch receives nextAutoHintTime, it resets the hint timer to show the hint at the updated time. Depending on how we generate on-demand hints, we may need a separate object to store automatic hints. For now, on-demand hints and automatic hints are generated by the same algorithm, and the only difference between them is timing.

4. **Hints UI**
   After receiving hints from the intelligent hint system, sage-node passes them to sage-scratch. Sage-scratch then updates the hints on the BlockPalette and resets the category and block hint timers (with nextAutoHintTime). When a timer goes off or the user asks for an on-demand hint, sage-scratch will shake the suggested categories or blocks. Additionally, we added a hint button for the user to request on-demand hints. When this hint button is clicked, the hints will be shown to the user immediately.

Limitations

1. The intelligent hint system requires a user type to generate hints. Currently, we have not yet integrated with the user behavior detection system, so the intelligent hint system uses a arbitrary user type.
2. We assume that the game snapshot will only change if there is a mouse click event. But this may change if we record more information than block names in the game snapshots or if more user interaction becomes available.

Publication

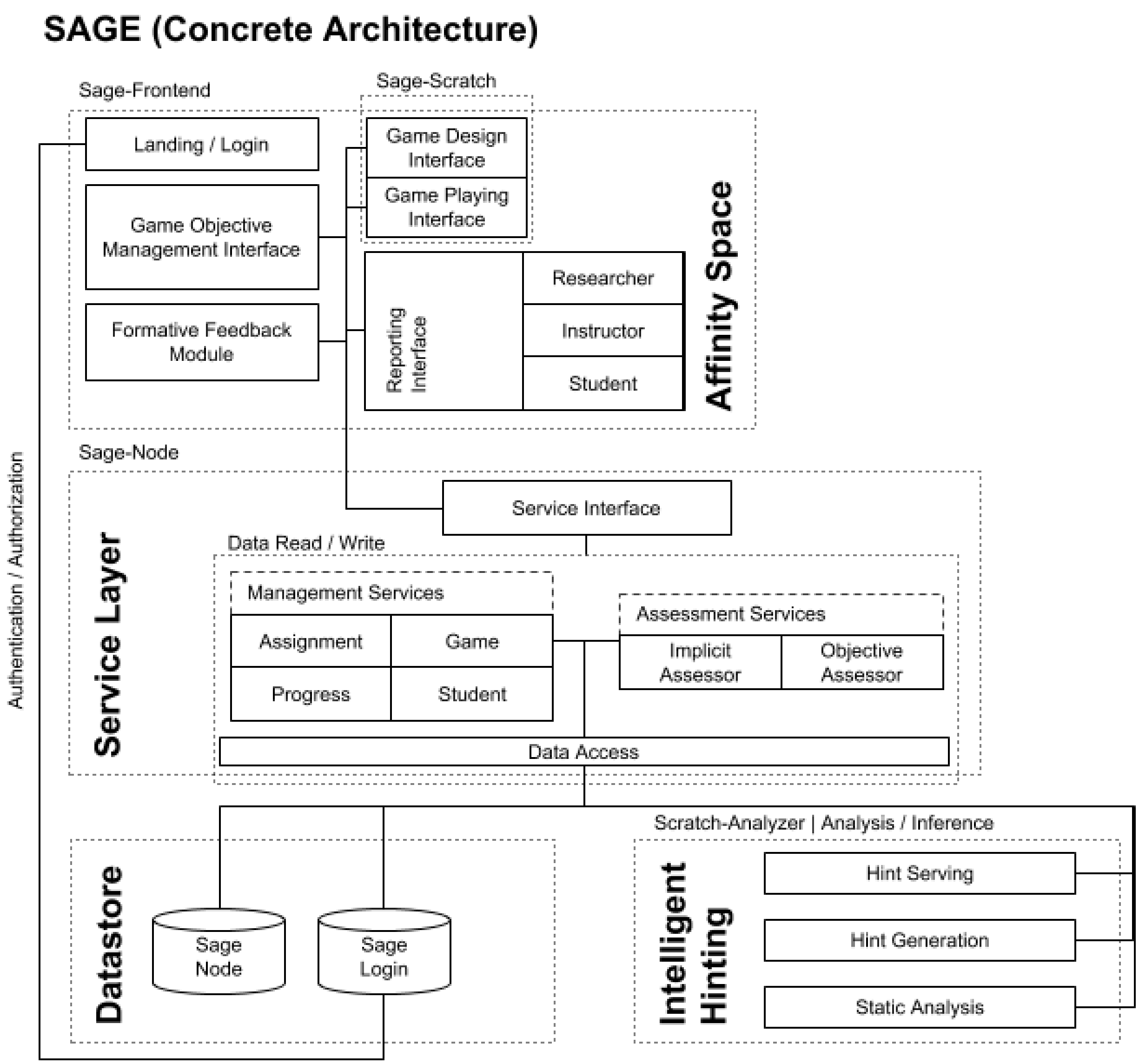
SAGE-RA: A Reference Architecture for Gameful Learning

SAGE-RA is a reference architecture that captures a set of fundamental subsystems, and the relationship between them, that is useful in the design and implementation of a learning environment with a consolidated user interface for constructionist and instructionist learning. We develop a reference architecture from our existing implementation and documentation (SAGE), and validate that architecture against code.org (Blockly-based), scratch.mit.edu(Scratch-based), and snap4arduino.rocks (Snap!-based). We 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.

Concrete Architecture

We developed SAGE's concrete architecture based on our experience with the project and analysis of the existing codebases. We placed a particular emphasis on Assessment and Formative Feedback, in particular as implemented via the Intelligent Hinting layer, as this is one of SAGE's most unique and defining features. This includes assessment of Game Objectives and learning paths.

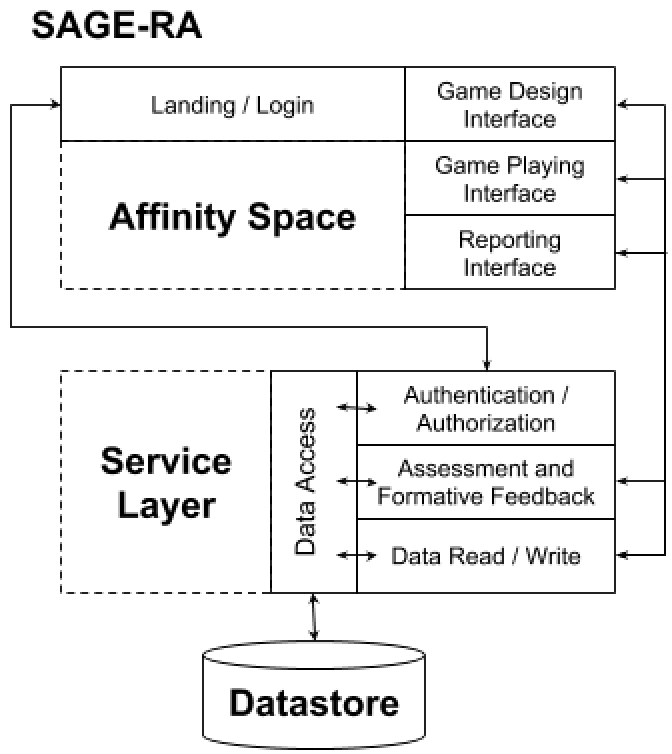
Using a process similar to Hassan and Holt's[16], we used this concrete architecture to derive a conceptual architecture, based on a generalization of the concrete architecture and plans for future development. From the conceptual architecture, we proposed a reference architecture, validated it against other implementations in the domain, and refined it.



Reference Architecture

In deriving SAGE-RA, we focused on its potential for application to a wide array of concrete learning environment software architectures. 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. The architecture could by implemented as a standalone application, a scalable distributed learning environment, a classic web application, or in other application models.

Assessment Flexibility

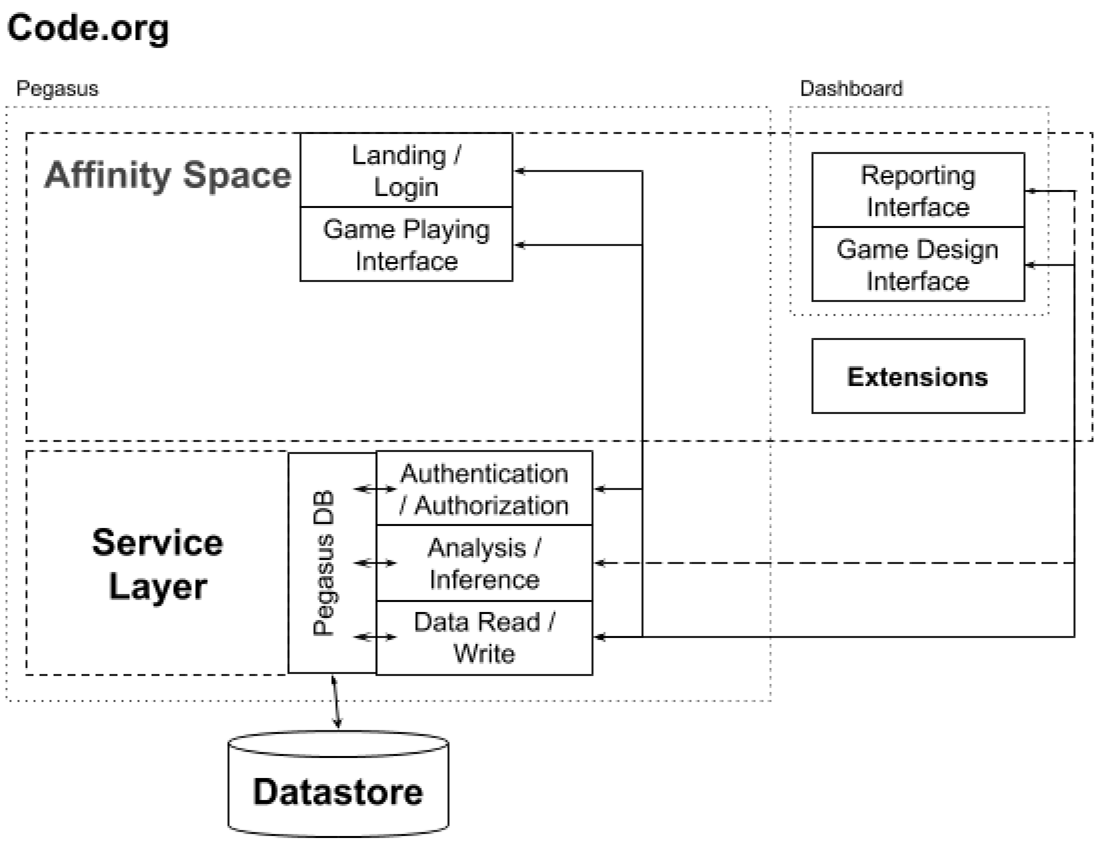
While SAGE's concrete architecture includes a Formative Feedback module and employs intelligent hinting to implement feedback based on student interactions, there is no prescriptive feedback model included in SAGE-RA. SAGE-RA does specify that feedback should be included and be reactive to assessment; the implementation of 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.

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 instructionist as constructionist game design, and could as easily be applied to games that integrate both approaches, or to future game-based learning methodologies.

Validation against code.org

We validated SAGE-RA against code.org's concrete implementation, which is available as a well-documented open source project. Code.org is a widely used learning environment in elementary and secondary schools, and powers Hour of Code which has engaged with 10% of all students in the world. We performed this validation by mapping code.org's primary subsystems to SAGE-RA.



Future Work: Next Steps

Our work this semester has surfaced a large amount of opportunities for additional work within DevOps, Gameful Intelligent Tutoring, and future SAGE publications. Next steps in each of these research areas include:

1. **DevOps** - A potential next step in DevOps would be to extend our random "fuzz" testing to other codebases outside of intelligent hinting. Fuzz testing against the sage-node API surfaces would be particularly useful, and may surface critical stability, flexibility, or latency issues. This fuzz testing could be included in our build automation, which would help prevent regressions, and quickly increase test coverage.
   As a concrete next step to be completed before the next semester begins, we will run automatic fixing of style errors in the sage-node codebase after final commits have been pushed by all teams.

2. **Gameful Intelligent Tutoring** - Integration with the user behavior detection system, would allow us to use concrete as opposed to arbitrary user types, and also allow us to analyze hint efficacy for different user types. Also, extending game snapshot capturing beyond mouse click events would allow us to capture more detailed user interactions (e.g. mouse movement or keystrokes), and may allow us to provide more nuanced hints.
3. **Publication** - An immediate next step for publication would be to complete the Future Work section of the rough draft of SAGE-RA. For the reference architecture to be useful and ready for publication, validation against other learning environments, including scratch.mit.edu and Blockly Games, is necessary.

Conclusion

Our work this semester focused on SAGE's code health and stability as the size and change frequency of the codebase increases, integrating, extending, and surfacing Gameful Intelligent Tutoring and Parson's Puzzle work from previous semesters, and preparing an initial draft of a reference architecture for future publication, or for use in components of other future SAGE publications.

Code-style checking and documentation[1], protection of mainline code branches through gated pull requests[2], a clean merge of Summer 2018 and Spring 2018's conflicting Parson's Puzzles work, and code coverage[3] checking of all primary SAGE codebases, comprised the bulk of our DevOps work, and should improve the time-to-productivity, and stability of SAGE through future semesters. SAGE-RA, includes our reference, conceptual, and concrete architectures for SAGE, and a validation against code.org's implementation[4]. Thoughtful implementation of an integration API for SAGE's existing intelligent hinting engine, and integration with the frontend interface and inference and analyses pipelines, has allowed us to surface Gameful Intelligent Hinting work done in this and past semesters within the SAGE UI, in a fully-integrated and extensible way, for the first time in the project's history.

References

[1] "Confluence." [Online]. Available: https://gudangdaya.atlassian.net/wiki/spaces/SAGE/pages/579141633/SAGE+Code+Standards. [Accessed: 17-Dec-2018]

[2] M. M. Rahman and C. K. Roy, "An Insight into the Pull Requests of GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, Hyderabad, India, 2014, pp. 364–367.

[3] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of DevOps," in *Agile Processes in Software Engineering and Extreme Programming*, vol. 212, C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds. Cham: Springer International Publishing, 2015, pp. 212–217.

[4] "SAGE-RA: A Reference Architecture for Gameful Learning." [Online]. Available: https://www.overleaf.com/project/5c01d470c432fe3bbde5eac5. [Accessed: 17-Dec-2018]

[5] 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.

[6] C. Murphy, G. Kaiser, and M. Arias, "Parameterizing random test data according to

equivalence classes," in *Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, 2007, pp. 38–41.
[7] S. Phillips, J. Sillito, and R. Walker, "Branching and merging: an investigation into current version control practices," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2011, pp. 9–15.
[8] V. Driessen, "A successful Git branching model," *URL http://nvie. com/posts/a-successful-git-branching-model*, 2010.
[9] L. Razzaq and N. T. Heffernan, "Hints: Is It Better to Give or Wait to Be Asked?," in *Intelligent Tutoring Systems*, vol. 6094, V. Aleven, J. Kay, and J. Mostow, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 349–358.
[10] 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*, 2015, pp. 195–204.
[11] Y. Ding, W. Luo, and J. Zhang, "Intelligent Hinting 1.1 Final Report," May 2018 [Online]. Available: https://github.com/cu-sage/Documents/blob/master/2018_1_Spring/Final_SAGE_git_intelligent_Hinting_1.1.pdf
[12] 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," *AAAI/IAAI*, Jul. 1999.
[13] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone, "The Concept of Reference Architectures," *Syst. Engin.*, vol. 2, 2009 [Online]. Available: http://doi.wiley.com/10.1002/sys.20129
[14] 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.
[15] E. Tufte and P. Graves-Morris, "The visual display of quantitative information.; 1983." 2014.
[16] 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.
[17] A. Grosskurth and M. W. Godfrey, "A reference architecture for web browsers," in *null*, 2005, pp. 661–664.
[18] "Confluence." [Online]. Available: https://gudangdaya.atlassian.net/wiki/spaces/SAGE/pages/579141633/SAGE+Code+Standards. [Accessed: 09-Nov-2018]
[19] "Flex SDK / Wiki / Coding Conventions." [Online]. Available: https://sourceforge.net/adobe/flexsdk/wiki/Coding%20Conventions/. [Accessed: 09-Nov-2018]
[20] "Google Java Style Guide." [Online]. Available: http://google.github.io/styleguide/javaguide.html. [Accessed: 10-Nov-2018]
[21] *standard*. Github [Online]. Available: https://github.com/standard/standard. [Accessed: 10-Nov-2018]
[22] G. van Rossum, B. Warsaw, and N. Coghlan, "PEP 8: style guide for Python code," *Python. org*, 2001 [Online]. Available: https://www.python.org/dev/peps/pep-0008/
[23] "The Puppet Language Style Guide," *Puppet*. [Online]. Available: https://puppet.com/docs/puppet/5.0/style_guide.html. [Accessed: 10-Nov-2018]
[24] V. Holen, *shellcheck*. Github [Online]. Available: https://github.com/koalaman/shellcheck. [Accessed: 10-Nov-2018]