Intelligent Hinting, DevOps and Publication

Alex Dziena

AGENDA

- Overview
- Dev Ops
- Publication
- Intelligent Hinting Integration

OVERVIEW

My focus this semester was on DevOps, specifically around creation of a pull request workflow and code health (code style and test coverage) tools, and integrating the work on Intelligent Hinting and Parson's Puzzles completed in Spring 2018 and Summer 2018, and on continuing SAGE-RA (reference architecture) preparation.

I believe the work done this semester stabilizes the entire SAGE codebase for future semesters, and sets a path to publication of a Reference Architecture.

DEV OPS: Integration

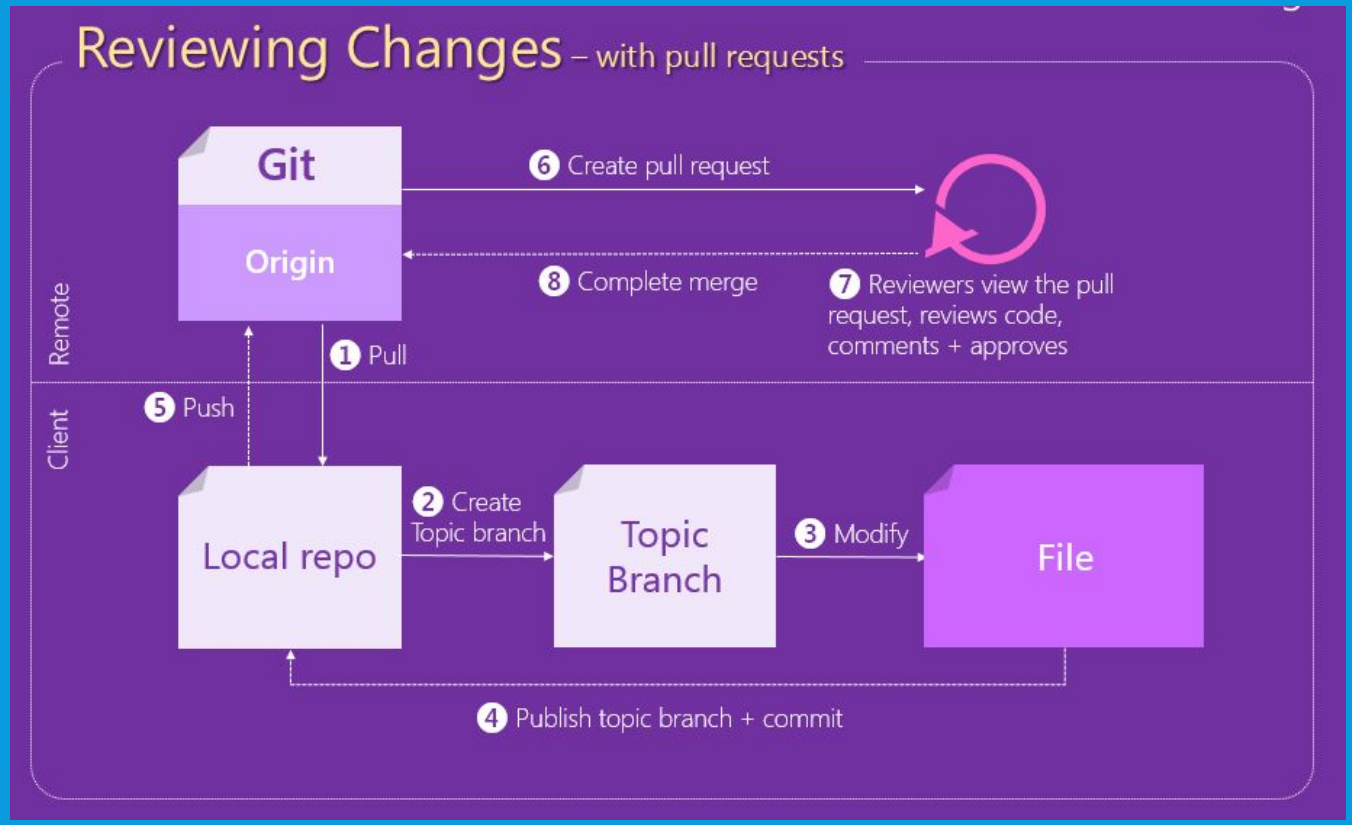
Breaking changes from Spring and Summer of 2018 were fully integrated this semester.

This enabled enhanced support for Parson's Puzzles without regressions in the codebase.
The complicated merge and integration process drove much of the work on code health and stability this semester.

DEV OPS: Pull Request Workflow

I've completed implementation of a pull request workflow. Commits that break the build on any of SAGE's codebases can not be commited to the primary development codebase.

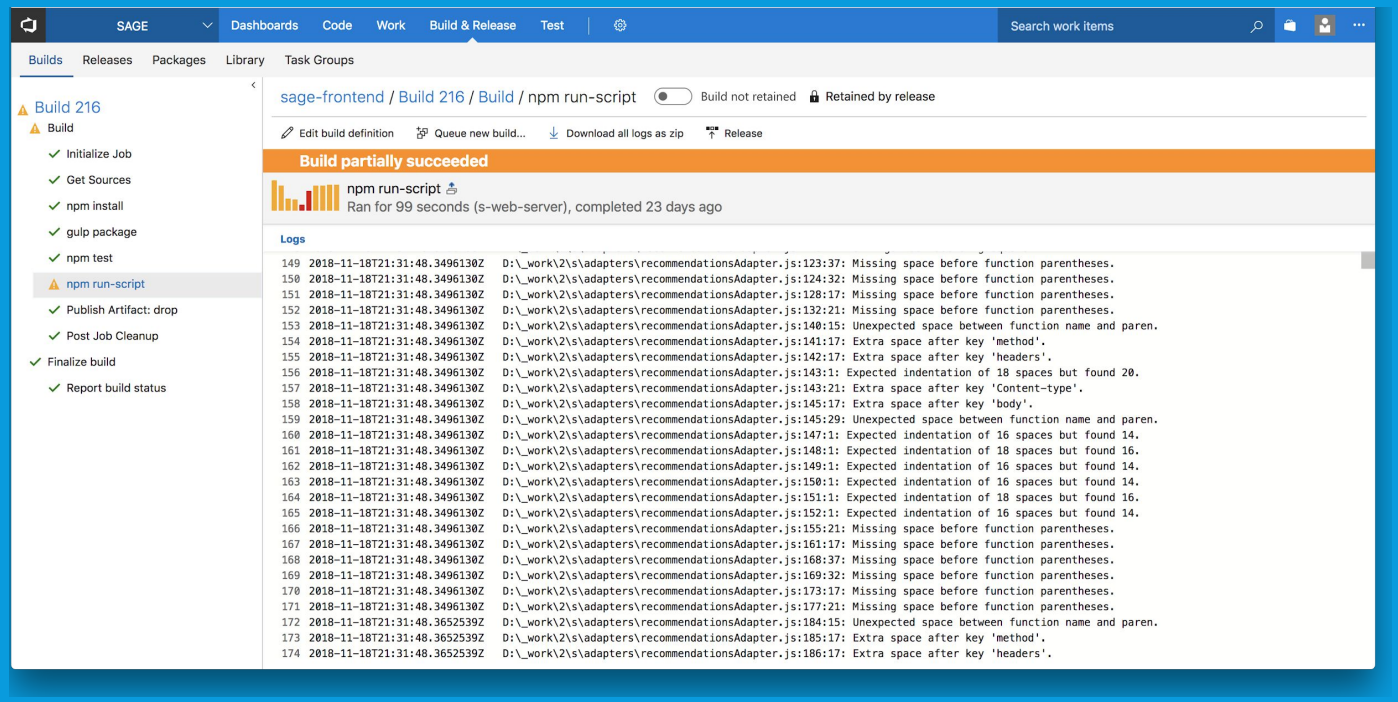
[1]



[1] Schaub, W. P. (2015, April 10). Git for the TFVC User – Workflow Investigations Part 2: Reviewing Changes (No Conflicts). Retrieved December 11, 2018, from https://blogs.msdn.microsoft.com/willy-peter_schaub/2015/04/10/git-for-the-tfvc-user-workflow-investigations-part-2-reviewing-changes-no-conflicts/

DEV OPS: Code Style and Test Coverage

- The sage-scratch, sage-frontend, and sage-node, scratch analyzer repositories all include non-build breaking code style checking.
- Build breaks on code style errors can be enabled with a configuration change.
- The scratch analyzer repository includes non-build breaking test coverage checking.
- Build breaks on test coverage regressions can be enabled with a configuration change.



DEV OPS: Summary

DevOps work completed this semester will improve SAGE's stability by forcing quality control on all incoming commits by:

- Ensuring all commits are build-able
- Enforcing code-style rules
- Enforcing test coverage of new commits

DEV OPS: Limitations and Future Work

- Test coverage and code style checks do not yet break the build. **Make these build-breaking after integration of this semester's work.**
- We have not yet hit the target of 30% test coverage. **Include this target in future semesters.**
- Testing API endpoints **remains** cumbersome. **Integrate Newman (Postman CLI via Node).**

PUBLICATION

- **"SAGE-RA: A Reference Architecture for Gameful Learning"** rough draft was completed this semester.

SAGE-RA: A Reference Architecture for Gameful Learning*

Alex Dziena, Jeffrey Bender, Gail Kaiser, and Lily Yu Li

***Abstract*— 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.**

I. INTRODUCTION

The value of reference architectures during the development of software systems is well documented [1]. Domain-based reference architectures[] exist for a large number of domains in engineering and education, but we have not found such an architecture within the domain of gameful computational thinking education. We will briefly define this domain, propose an architecture for use in developing constructionist and instructionist learning environments within that domain, validate that architecture against several widely-used web-based learning environments, and present a roadmap for future work.

II. THE GAMEFUL COMPUTATIONAL THINKING EDUCATION DOMAIN

A. Computational Thinking

Add computational thinking summary

B. Game-based learning

Researchers began exploring the potential of video games for learning in the early 1960s [2], but GBL received less traction in academia prior to Gee's 2003 treatise [3] which illuminates similarities between textbooks and game manuals, and highlights the lucidity with which players can understand the text within a manual after gameplay. Subsequent studies demonstrate the appropriation of curricula content as tools for successful gameplay in classrooms [4], and the design of games as pedagogical playscapes for students to think like professionals [5], while more recent work popularizes characteristics of gameful experience: goal-oriented, curiosity-driven, failure-fearless, optimistic, and fun [6]. This pioneering has normalized perceptions of the medium [7], and led to acceptance in schools, especially when game design is grounded in theories from the learning sciences [8]. The CS community has pioneered, as well, especially in employing game creation as an activity that applies curricular content [9]. Research that gamifies the software development process [10] has led to achievement systems within integrated development environments [11], but the increasing prevalence of games that teach CS concepts is comparatively recent [12], [13]. While many game design patterns exist [14], two frameworks emerge that deserve additional attention: Parson's Programming Puzzles (PPP) [15] and Constructionist Video Games (CVG) [16].

PPP enable students to practice CT by assembling into correct order sets of mixed-up blocks that comprise samples of well-written code which focus on individual concepts. For example, a PPP might contain a mixed-up set of blocks necessary to perform a swap of values between two variables as a means of introducing a sorting algorithm. Due to the small scope of each puzzle, the teacher can impart good programming practice by offering accompanying direct instruction [17]. This approach has produced promising results which indicate PPP can reduce grading time and variability [18], increase learning efficiency [19], and facilitate online classification of problem solving strategies so that formative feedback can activate when needed, and continuous improvement to the puzzle corpus and learning environment can ensue [20].

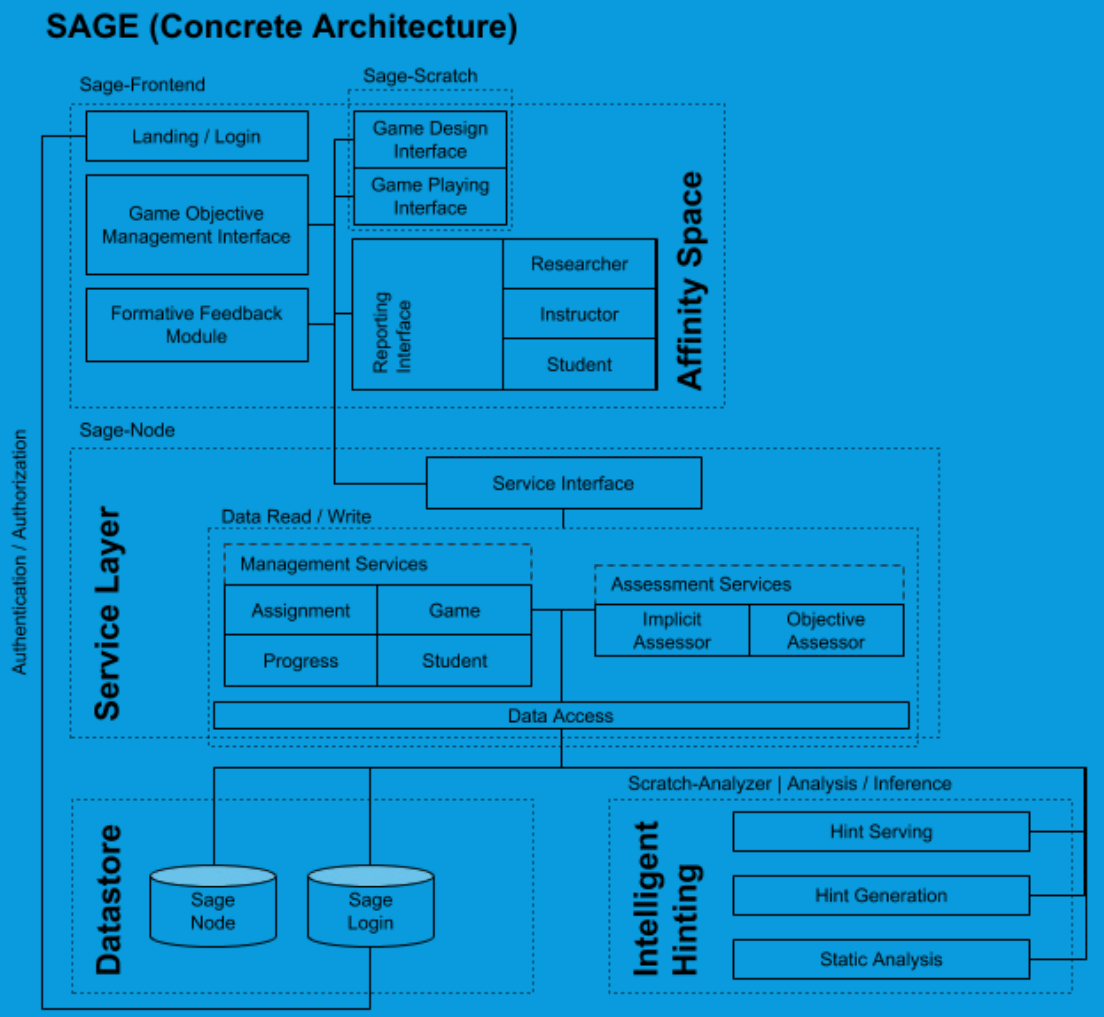
CVG enable students to explore CT using construction tools sufficiently expressive for personally meaningful gameplay. For example, a CVG might allow the use of any blocks, but require the sorting of in-game objects using limited block-point resources. Teachers scaffold learning by defining incremental goals [21], while affording students the opportunity to take ownership of the experience and progress through the sequence of interest and motivation toward sustained engagement [22]. When strategically arranged within a learning progression after PPP gameplay produces evidence of comprehension, CVG could amplify the impact of direct instruction by providing the sculpted context in which students can apply CT concepts, thereby broadening and deepening understanding. The classroom potential of this gameful structure is one area the results of our study address in Section 4.2

III. DERIVING SAGE-RA

Using a process similar to Hassan and Holt's[23], 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 other implementations in the domain, and refined it.

PUBLICATION: Concrete Architecture

This draft includes a diagram of SAGE's Concrete Architecture, which was used to derive the reference architecture, SAGE-RA.

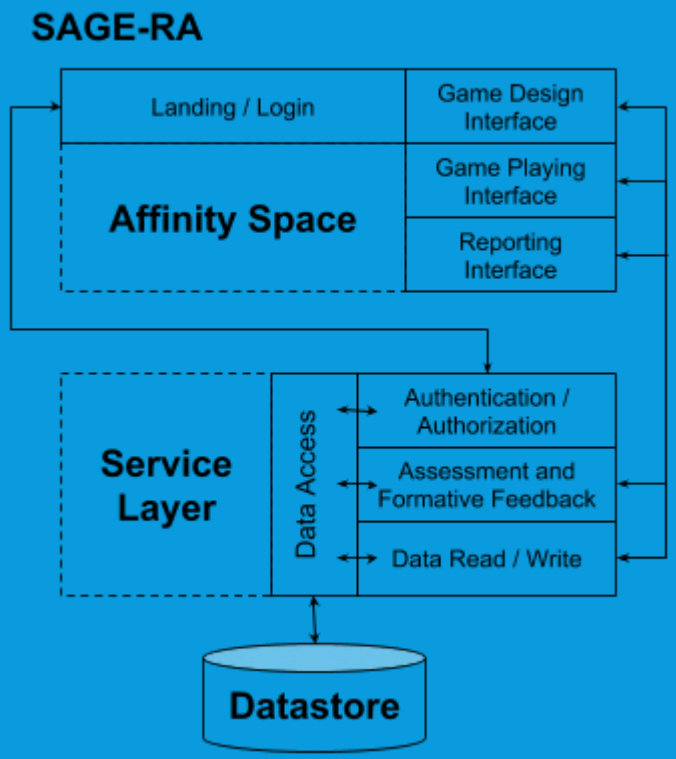


PUBLICATION: SAGE-RA

SAGE-RA is combined Angelov Type 3.1 and Type 5.1 reference architecture[1] for the Gameful Computational Thinking Education domain. It features

- Client-Independence
- Assessment Flexibility
- Game Design Flexibility

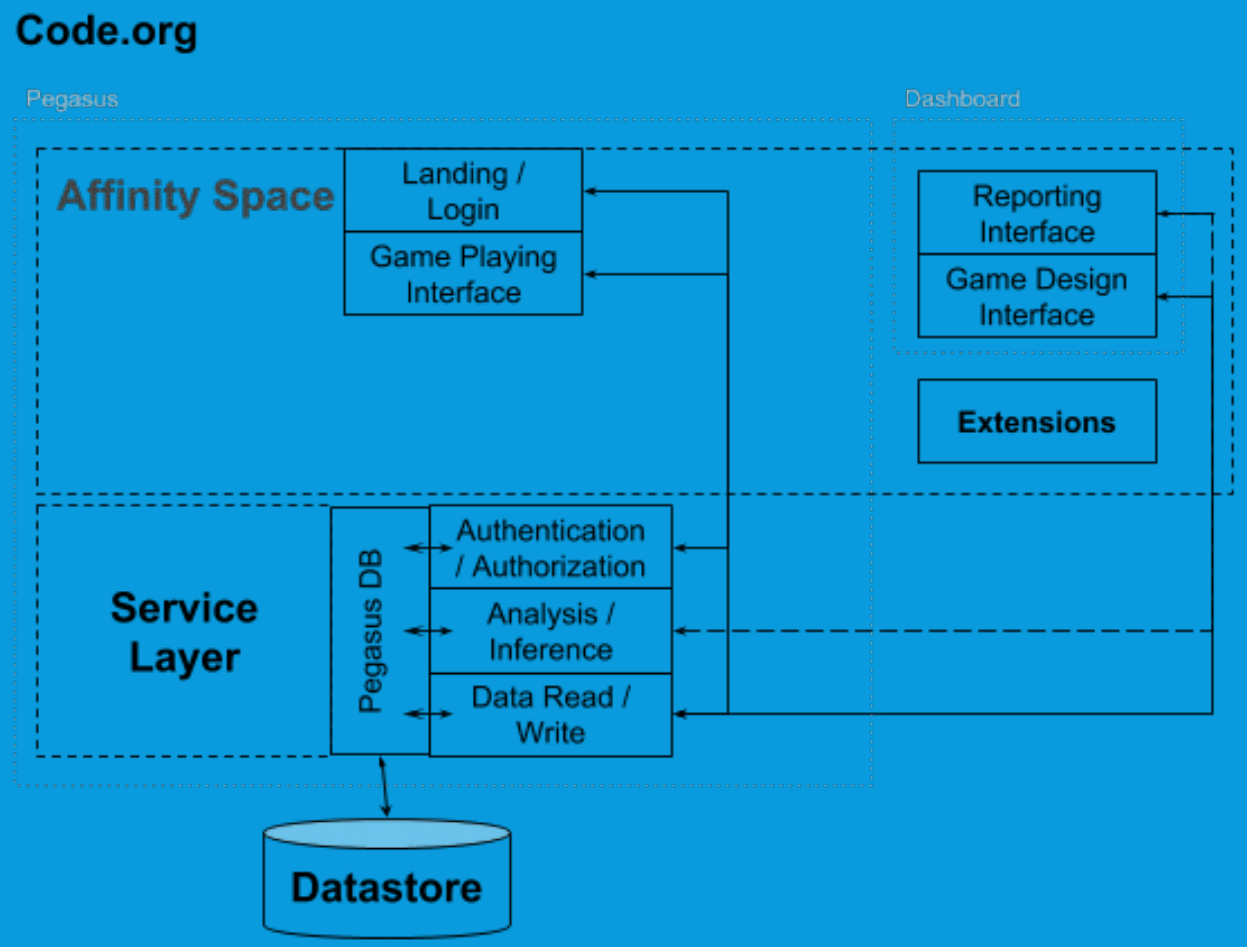
and is validated against code.org, scratch.mit.edu, and Blockly Games.



[1] Angelov, Samuil, Paul Grefen, and Danny Greefhorst. "A framework for analysis and design of software reference architectures." Information and Software Technology 54.4 (2012): 417-431.

PUBLICATION: Code.org Validation

SAGE-RA provides a close mapping to Code.org's architecture, with all Code.org modules mapping to a layer of SAGE-RA.



PUBLICATION: Future Work

The reference architecture is currently in rough draft form, with some sections only outlined. **Future work next semester could include finishing the reference architecture and preparing it for submission.**

Further validation of the reference architecture is needed. **Future work could include identifying other gameful learning environments and validating the architecture against them, as outlined in the rough draft.**

1) Angelov, Samuil, Paul Grefen, and Danny Greefhorst. "A framework for analysis and design of software reference architectures." Information and Software Technology 54.4 (2012): 417-431.

INTELLIGENT HINTING INTEGRATION

Intelligent Hinting Integration with the NodeJS application creation of a hint retrieval endpoint, and integration of this API with the sage-scratch swf to make hint data available to the UI is covered in the GIT team's presentation.

A chatbot question classification prototype has been started, but the model needs to be retrained on a new question corpus. UI integration won't be feasible this semester.