Formative SAGE Assessments: Final Report

Jairo Pava

COMS E6901, Section 14

Spring 2016

TABLE OF CONTENTS

1.0 Introduction

This report describes the Formative SAGE Assessments project. The objectives of this project are to 1) Provide teachers with the ability to automate assessment of SAGE (Bender, 2015) assignment submissions and 2) Provide students with real-time guidance as they work on their SAGE assignments. Both objectives were chosen to facilitate computational thinking (Barr & Stephenson, 2011) instruction and learning.

Computational thinking is an approach to solving problems using concepts of abstraction, recursion, and iteration. Mastery of these concepts fosters critical thinking skills that are not just important in computer science but are crucial for the academic and personal development of young children through their adult lives. As such, this problem solving methodology can be transferred and applied across subjects in the classroom. Teaching computational thinking, however, has its challenges. Many teachers do not have the training to lead classroom discussions or prepare assignments to build this skill in their students. And even when teachers are motivated, sometimes it is a lot harder to keep students engaged because these skills can be difficult and frustrating to grasp at first.

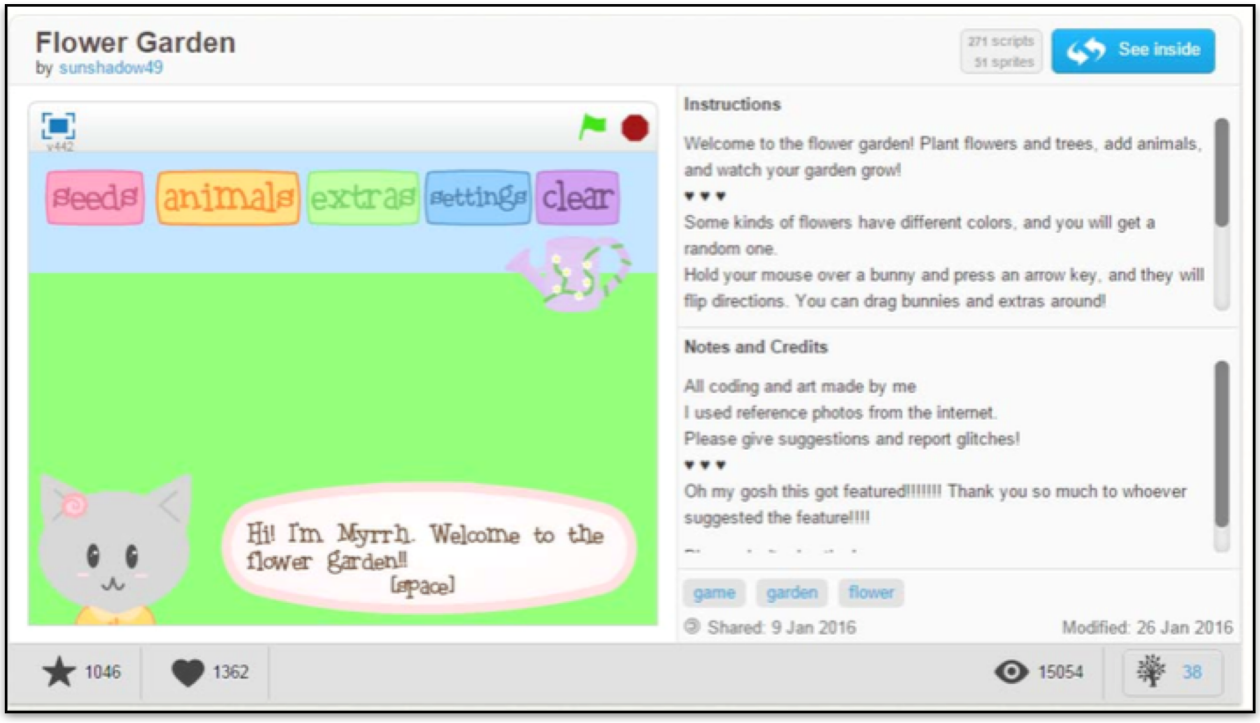

Figure 1 Scratch Project

Research in computer science, education, and the marriage between the two have led to software tools that address the challenges of teaching computational thinking in the classroom. The Lifelong Kindergarten Group at the Massachusetts Institute of Technology provides a great example of this with Scratch (MIT Lifelong Kindergarten Group, 2016). Scratch is a programming language and online community where anyone can program and share interactive media such as stories, games, and animations with people from all over the world. The programming language is drag-and-drop based and programs, like the one shown in Figure 1, consist of two-dimensional, interactive animations. As of May 2016, about 9 million registered users have shared more than 12 million projects on the Scratch website. The majority of those users are between the ages of 8 and 16.

Millions of people are creating Scratch projects in a wide variety of settings including homes, schools, museums, libraries, and community centers. It is easy to learn. It engages children. And it is a powerful learning tool.

SAGE extends Scratch by empowering teachers to play the role of game designers. This enables them to create games for students to play as they learn computational thinking concepts. Since games are fun and engaging for young students, SAGE leverages game-based learning techniques to instill intrinsic motivation for computational thinking at an early age. The teacher designs games and students play the game by building Scratch programs to solve objectives.

SAGE takes advantage of the collaborative nature of the online Scratch community and traditional classroom environment to encourage students to compete with each other while playing the games, and most importantly, learning computational thinking concepts. However, it is ultimately up to the teacher to provide a final evaluation of the student's completion of the game.

This evaluation is time consuming and does not scale well. Evaluating student submissions of completed games is both subjective and objective. Objective criteria are used to determine whether the student completed specific, measureable tasks laid out in the game objectives. Subjective criteria, on the other hand, include the aggregate attention to details by students in their work that only a human can perceive but make the clear difference between students that have understood a concept versus students that have *mastered* a concept. Evaluating subjective criteria is the most creative aspect of a teacher's work and meaningful feedback in this aspect will immensely benefit a student's progress.

The Formative SAGE Assessments project enables teachers to automate objective evaluations of student submissions. These evaluations automatically provide real-time feedback to students as they work on their SAGE assignment. As a result, students will know how close they are to completing their game and whether they are on the right path, very much like popular games today track progress towards completion. These automated evaluations will also free up teacher time to focus on the subjective evaluation of student submissions.

2.0 Related Work

Although there exists prior work that provides students with feedback as they complete programming assignments, no prior work was found that leverages that feedback within the context of Scratch to facilitate computational thinking instruction. The sections below describe related work and how those ideas have been extended.

2.1 Retina

Retina (Murphy, Kaiser, Loveland, & Hasan, 2009) is a tool that collects information about students as they complete computer programming assignments. This information includes time spent on the assignment, number of successful and failed compilations, among other related data. The course professor and student are then provided with reports by Retina based on the aggregation of that information across all students in the course. An example of this report is illustrated in Figure 2.

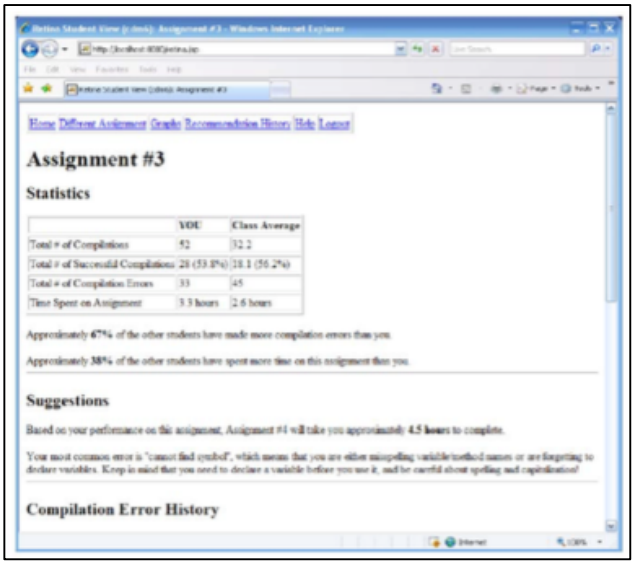

Figure 2 Retina Assignment Feedback

Retina is able to generate personalized suggestions to students as they complete their computer programming assignments to guide them towards successful completion. Anecdotal evidence

demonstrated Retina's utility in helping course instructors better plan class and one-on-one discussions with students.

The Formative SAGE Assessment project extends the ideas from Retina by providing personalized feedback in the way of block restrictions. For example, while Retina will provide a message with personalized feedback, the SAGE Assessment project will enable or disable the blocks a student may use to work on their assignment. By restricting the blocks a student may use, she will be more likely to focus on the concepts that the teacher intends for her to learn. Furthermore, The Formative SAGE Assessment project will record the combination of feedback it provides a student and the subsequent actions the student takes. This recorded data will enable teachers to understand the impact their suggestions have on student work and enable them to make changes to the recommendations, if necessary.

2.2 HAIRBALL

Hairball (Boe, et al., 2013) is a static analysis tool for Scratch project files. The tool receives saved Scratch project files as input and produces an analysis of the project. The analysis is performed by a collection of Hairball plugins that work independently to analyze the project based on different criteria. Currently, Hairball has plugins that analyze the implementation of Scratch programs for competence in initialization, broadcast and receive, say and sound synchronization, and animation.

Hairball allows for the easy development of plugins using Python to add to the analysis of Scratch project files. Figure 3 illustrates an example Hairball plugin that keeps track of the number of unique blocks in a project.

```
class BlockCounts(HairballPlugin):
  def analyze(self, scratch):
    blocks = Counter()
    for block, _, _ in iter_blocks(scratch):
      blocks.update({block: 1})
    return blocks
```

Figure 3 Hairball Plugin Example

Just like Hairball, the Formative SAGE Assessment project is extensible via plugins. These plugins may be used to extend the types of assessments that may be used to evaluate student SAGE submissions. The plugin system was inspired by Hairball but differs in its implementation. Hairball plugins must be written in Python. This requires at least an intermediate level of understanding of the Python language. The Formative SAGE Assessment project, however, can be extended by

registering HTTP endpoints that are called based on various configurable triggers. In this manner, plugins may be written using any programming language of choice as long as it implements a contract defined by the Formative SAGE Assessment project. This flexibility allows anyone to quickly write plugins.

2.3 Dr. Scratch

Dr. Scratch (Moreno-Leon, Robles, & Roman-Gonzalez, 2015) is a web application where users upload Scratch project files for automated analysis. After a project file is analyzed, a scorecard is presented that indicates how well the project uses a variety of computational thinking concepts. An example of this scorecard is illustrated in Figure 4. Dr. Scratch has two goals: to support educators in the assessment of student projects and to encourage students to keep improving their programming skills. Feedback from Dr. Scratch enables students to understand that successful completion of an assignment includes more than just completing a set of tasks. Successful completion also includes mastering computational thinking concepts that improve their ability to complete similar assignments in the future even if they are not in the same domain.

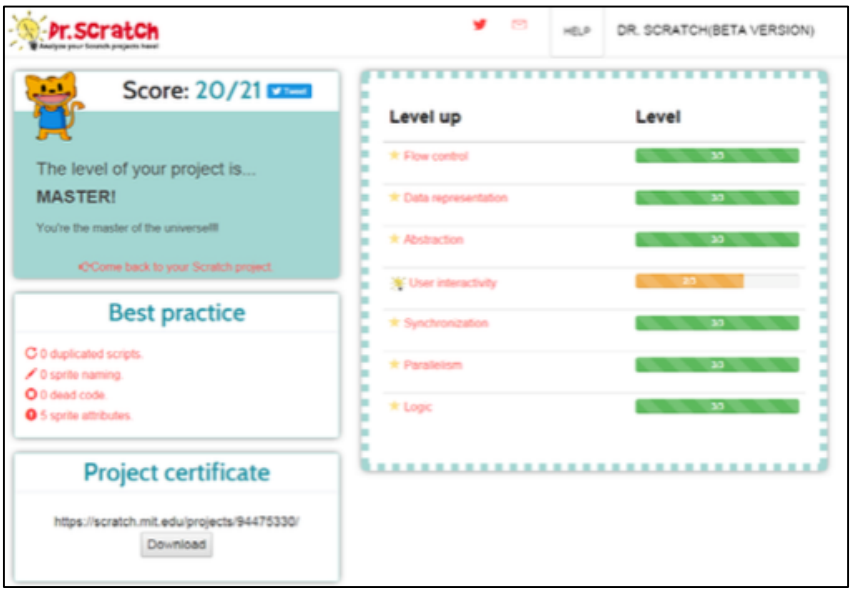

Figure 4 Dr. Scratch Project Evaluation

The Formative SAGE Assessment project differs from Dr. Scratch in several ways. Unlike Dr. Scratch, projects are automatically and periodically uploaded, without student intervention, from the Scratch editor to the assessment server for evaluation. This allows students to receive feedback as they work on their projects rather than at the end. Furthermore, the communication

between the Scratch editor and the assessment server is two-way. Once the assessment server evaluates a student's project, it may influence which blocks are restricted in the Scratch editor. This restriction allows a student to focus on particular computational thinking concepts that the instructor wants to target. Finally, the criteria that Dr. Scratch uses to evaluate evidence of computational thinking in submitted projects is rather simple. The criteria behind this evaluation is solely based on the presence of specific blocks. Much more sophisticated models have since been presented to the research community, like PECT (Seiter & Foreman, 2013), that define and analyze design patterns to find evidence of computational thinking. The Formative SAGE Assessment project provides instructors with a visual assessment language that enables the automation of computational thinking analysis like that proposed by PECT.

3.0 Features

This section will describe the features of the SAGE Assessment Project at a high-level for a general understanding of how it is meant to be used in the classroom.

3.1 Visual Assessment Language

The visual assessment language is a drag-and-drop based language that allows teachers and students to write assessments for SAGE assignments. Teachers may use the language to write assessments that guide students as they work on their assignments. Students may use the language to practice a Test-Driven approach towards the development of their assignments.

Figure 5 demonstrates an example of an assessment that may be created. Notice that the blocks are similar to the blocks used to build Scratch programs. The similarity is intentional. The American Educational Research Association prescribes curriculum design that encourages teachers to develop strong subject matter knowledge on the topics presented to students (Davis & Krajcik, 2005). By using a language that is similar to the language students will be using to work on their assignments, teachers will better connect theory with practice.

The language allows for the creation of assessments that verify presence of sprites, sprite interactions with other sprites, sprite interactions with the stage, and sprite responses to keyboard key presses. The types of assessments that the language currently supports was motivated the Creative Computing Curriculum from the Harvard Graduate School of Education (Brennan, Balch, & Chung, 2015). The curriculum defines a diverse set of Scratch programming assignments that

target a variety of computational thinking concepts. It defines the assignment that students are to complete and a list of assessments that the teacher may use to evaluate student submissions of that assignment. The Visual Testing Language enables automated assessment support for most of the assignments in the chapter on Games.

Multiple triggers may also be added to an assessment that execute an action when the assessment passes or fails. An action may include the ability to output a message on the dashboard that the student can view while working on the SAGE assignment. An action may also include the ability to enable or disable a student's access to blocks while working on the SAGE assignment. The goal behind the triggers is to enable the instructor to guide the student towards a particular programming concept.

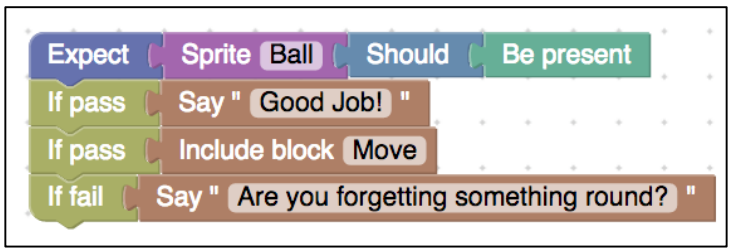

Figure 5 Example Assessment

The language may also be used to verify the presence of block and block design patterns, as prescribed by the PECT model (Seiter & Foreman, 2013), to identify the degree of computational thinking demonstrated by a student's assignment. The PECT model proposes a way to map the presence of Scratch blocks and a combined use of those blocks to categorize student's use of the following CSTA computational thinking concepts (Barr & Stephenson, 2011):

- Procedures and Algorithms
- Problem Decomposition
- Parallelization and Synchronization
- Abstraction
- Data Representation

These computational thinking concepts are categorized as follows:

- Basic
- Developing
- Proficient

A visual assessment language editor was implemented to allow users to create assessments. Figure 6 presents a screen capture of the editor. The main layout of the editor consists of two columns. The left-most column displays links to various color-coded assessment block types. When a link is

clicked, the column expands to provide the user with specific blocks that may be selected to create the assessment. Blocks may be selected from this column and dragged onto the second column where they may be connected with other blocks to complete the assessment. Blocks will only connect to other blocks that adhere to the grammar of the language.

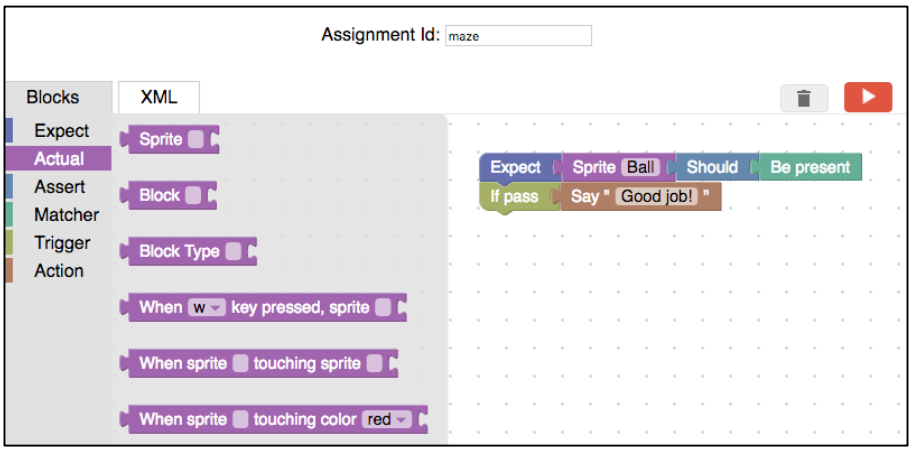

Figure 6 Visual Assessment Language Editor

Once a user finishes connecting blocks to create an assessment, he may click on the tab labeled "XML". This tab will display an XML version of the assessment language. A user may modify the XML displayed under this tab to update the blocks. An example use case of this would be to load XML from a previous editing session.

A user may also upload the assessments by entering an assignment ID in the text box labeled "Assignment Id" and clicking on the red play button. The XML version of the blocks is then uploaded to the assessment server that will later be used to evaluate SAGE assignment submissions.

3.2 Scratch Editor

The Scratch Editor is used by students via a web browser with Flash (Adobe, 2016) support. With the editor, students may implement Scratch assignments by creating graphics, called sprites, and associating scripts to those sprites for animation. The scripts are created by dragging and dropping blocks onto the editor. The blocks represent commands and they may be chained together to form complex scripts. Figure 7 presents a screen capture of the editor that demonstrates three columns with sprites on the lower left, blocks in the middle, and scripts on the right.

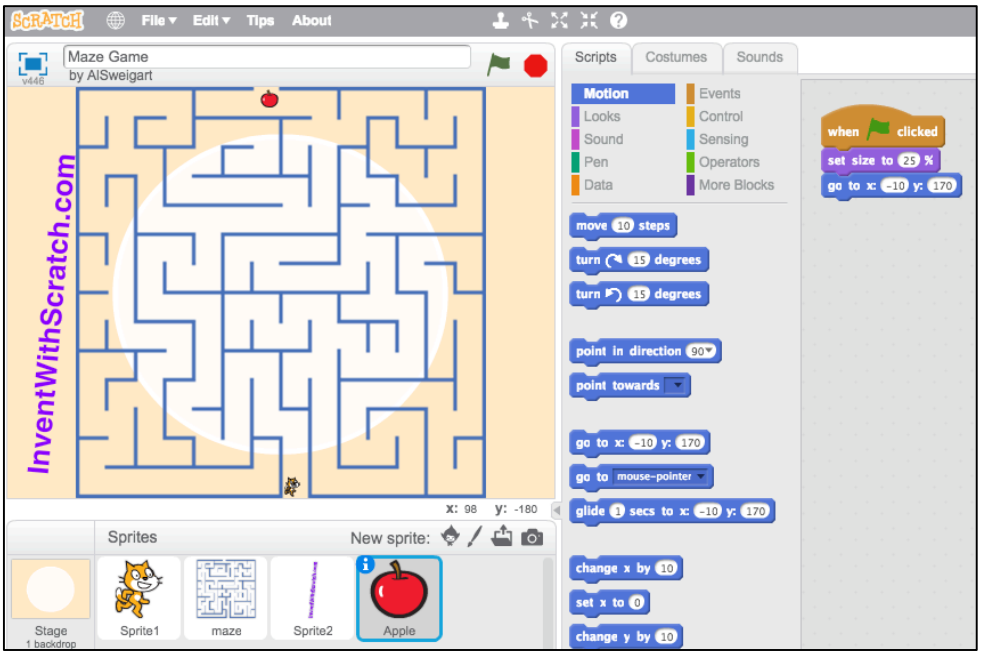

Figure 7 Scratch Editor

The Scratch editor was customized by the Formative SAGE Assessment project in several ways. When a student opens the editor for the first time, she is prompted for her student ID and the assignment ID, as illustrated in Figure 8. If the student enters this information, then the Scratch editor will periodically save the student's work and upload it to the assessment server. This happens in the background without disrupting the student's work.

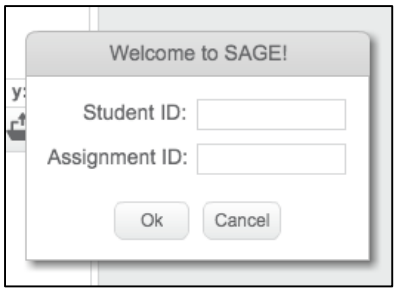

Figure 8 Student Prompt

When the project is uploaded to the assessment server, the server will reply with an assessment. This assessment is based off the assessments that were uploaded using the visual assessment language editor and the assignment ID that corresponds to the ID that the student entered when prompted earlier. The scratch editor will parse the assessment and enable or disable blocks, if that

is an action that should be triggered. These block restrictions extend the work by Jeff Bender (Bender, 2015) to provide instructors with a way to guide students towards particular concepts that the instructor wants them to focus on.

3.3 Dashboard

Figure 9 presents a screen capture of the student dashboard. This dashboard is loaded in a separate web browser window while the student uses the Scratch editor to work on a SAGE assignment.

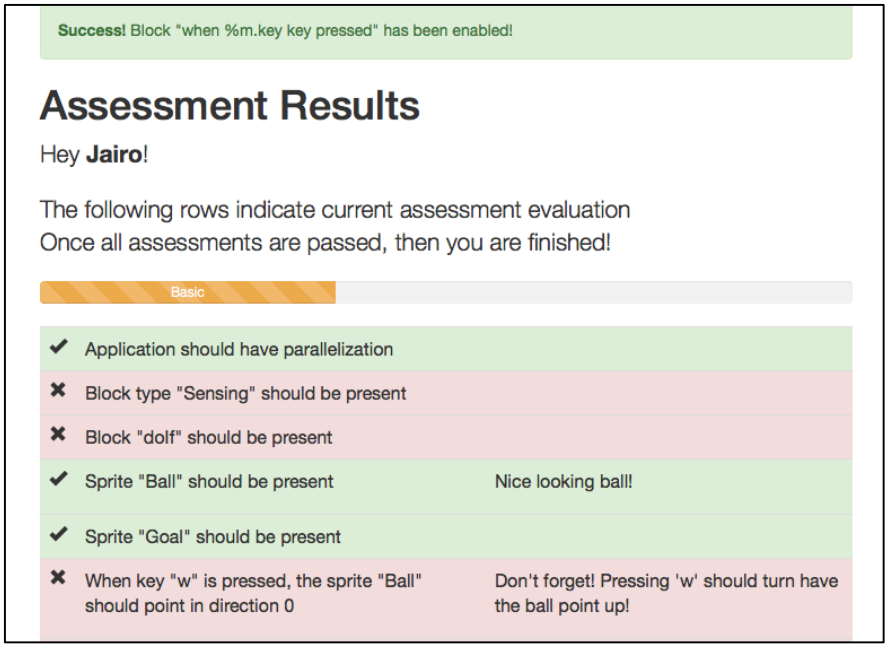

Figure 9 Student Dashboard

The dashboard is automatically updated when the Scratch editor uploads the student's work to the assessment server. It indicates how close the student is to completing the SAGE assignment by color coding and checking off completed assessments. It also displays feedback that the teacher may provide to the student via the triggers in the visual assessment language. An animated progress bar indicates to the student how proficient they are based on the assessments created by the teacher. Finally, when the student unlocks new blocks by completing an assessment, an alert is displayed on the top of the dashboard to celebrate the student's accomplishment. This alert fades away after a certain time.

4.0 ARCHITECTURE

Figure 10 presents a diagram of the architecture for the Formative SAGE Assessment Project. The rectangles represent standalone applications. The arrows between the applications represent HTTP requests. The cylinder represents data storage. Requests to this storage are in a protocol specific to the storage mechanism. The arrowheads represent the source and target of the requests.

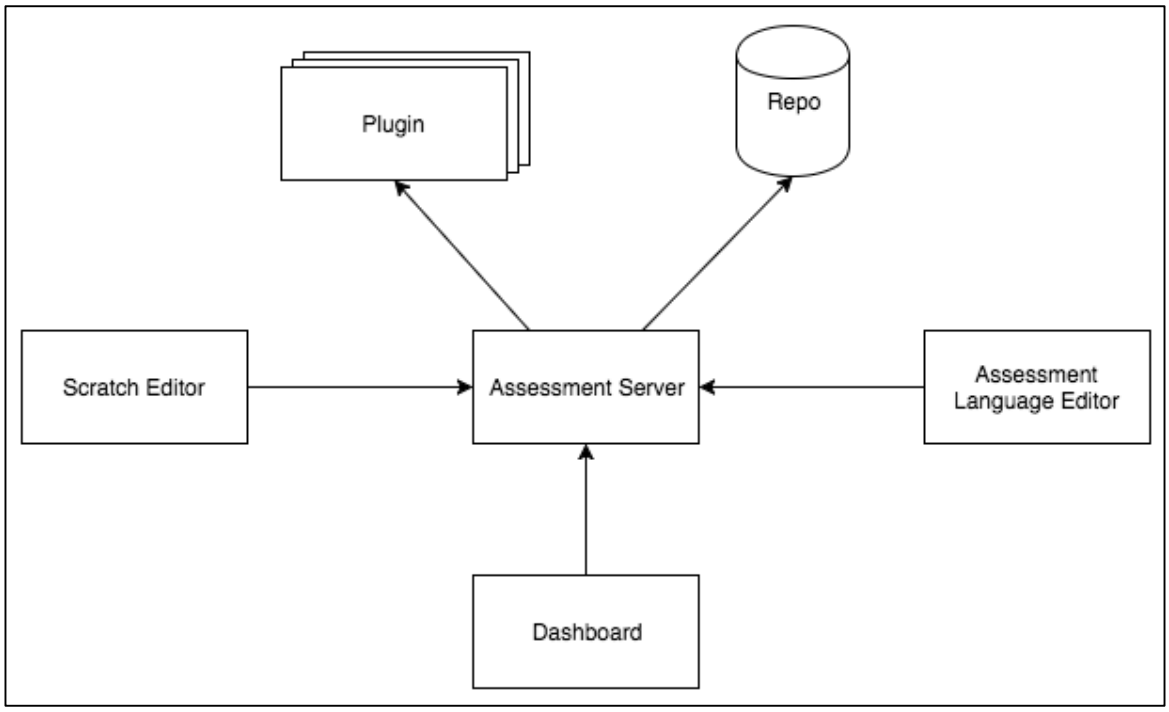

Figure 10 Architecture

At the center of the diagram is the Assessment Server. All applications communicate with this server. It is responsible for receiving and managing the storage of student assignments and assessments. It is also responsible for executing the assessments against the assignments and generating an assessment result. The server leverages a data store repository to maintain records of assignments, assessments, and assessment results. The server is also extensible via plugins. Plugins are stand-alone applications that register HTTP endpoints on the assessment server and are called whenever a configurable trigger is invoked.

The Assessment Language Editor POSTs XML-encoded assessments to the Assessment Server. Users of the editor will create assessments and click a button to POST the assessment and receive confirmation from the Assessment Server.

Students will use the Scratch Editor to work on SAGE assignments. The editor will periodically upload the JSON-encoded assignment to the Assessment Server while the student is working. The Assessment Server will respond with JSON-encoded assessment results that the Scratch Editor will interpret to include or exclude blocks for the student to use.

The Dashboard will periodically poll the Assessment Server to retrieve the latest assessment results for a student. The dashboard will represent the assessment results to users via a web browser.

5.0 Implementation

This section will describe implementation details of the Formative SAGE Assessment project including specific technologies, designs, and patterns used.

5.1 Assessment Server

The Assessment Server was implemented using the Go programming language from Google (Google, 2016). Go was chosen primarily because it is a language that the author of this report was already familiar with and could use to start implementing the server immediately. The server is responsible for managing the storage of assignments, assessments, and assessment results. It is also responsible for executing assessments against assignments and evaluating the assessment results.

5.1.1 HTTP INTERFACE

The server exposes a RESTful HTTP interface for consuming applications. The HTTP endpoints are documented below:

Description	Upload an assessment with the given ID
Route	POST /assessments/{id}
Request Body	XML-encoded visual assessment language blocks
Response Code	204, on success 500, on other error
Response Body	None

Description	Upload an assignment with a given ID from a student with a given ID
Route	POST /students/{id}/assignments{id}
Request Body	JSON-encoded Scratch project
Response Code	204, on success 404, if assignment or student id not found 500, on other error
Response Body	None

Description	Retrieve assessment result with a given ID from a student with a given ID
Routes	GET /students/{id}/assessments/{id}/results
Request Body	None
Response Code	200, on success 404, if assessment results or student ID not found 500, on other error
Response Body	{ [{ “Description”: “…”, “Pass”: true, “Actions”: [{ “Type”: “…”,

```
                                                    "Command": "..."
                                                }
                                            ]
                                        }
                                    ]
                                }
```

5.1.2 Storage

The Assessment Server uses MongoDB (MongoDB, Inc., 2016) for storage of assignments, assessments, and assessment results. MongoDB is a document-oriented database that fits well for its current use case. Documents are stored in three separate collections and contain no relational meta-data. The server references them individually by an ID supplied by the consumer of the HTTP API. MongoDB is easy to deploy and facilitated quick development of the server.

5.1.3 Assessment Evaluation

When the Assessment Server receives an assignment from the Scratch Editor or an assessment from the Visual Assessment Language editor, it parses those documents and stores them in a data structure that will facilitate assessment evaluation later on. For instance, Scratch assignments are stored as a list of sprite objects which themselves consist of a list of script objects. The script objects contain a string representation of the blocks that belong to that script. Likewise, Assessments are stored as a tree of block objects. Each block object has a reference to zero or more blocks that extend vertically or horizontally from it. Each block object contains information about the block type and the data that is pertinent to that block. The tree as a whole represents the assessment and each block within it contains information necessary to execute that assessment against an assignment.

When the Assessment Server receives a request to execute an assessment against an assignment, it retrieves both documents from the database. For every assessment, the server iterates through the blocks in the tree to interpret what the assessment intends to verify. The server then iterates through the list of sprites to perform a static analysis of the scripts within those sprites and evaluate whether the assessment should pass or fail.

After the server determines which assessments have passed or failed, it parses the zero or more triggers that may be attached to each assessment. The result is a JSON array consisting of a JSON object per assessment that describes the assessment, whether it passed or failed, and a JSON array of triggers that should be invoked based on the result of the assessment.

5.1.4 PLUGINS

To facilitate future work on the Assessment Server, it was designed for extensibility via a plugin system. Plugins may be used to override the Assessment Server's default behavior for running an assessment. To enable a plugin, a configuration file must be provided to the Assessment Server that registers an HTTP endpoint to an assessment type. The server will then call that HTTP endpoint when running an assessment of that type instead of evaluating the assessment itself and use the result of the HTTP response as the evaluation of the assessment. To work correctly, plugins must adhere to an API contract for the incoming HTTP request and outgoing HTTP response.

For example, if a separate plugin was written to handle assessments that verify the presence of sprites, then the following steps must be taken. First, the *plugins.json* file in the root directory of the Assessment Server must contain the following entry:

```
{
        [
                {
                        "Type": "actual_sprite",
                        "Handler": "http://localhost/sprite"
                }
        ]
}
```

This JSON Array contains one JSON Object that tells the Assessment Server that any assessment of type "actual_sprite" should defer its evaluation to the HTTP endpoint at "http://localhost/sprite". Second, the server implementing the plugin should expose an endpoint at the /sprite route that accepts HTTP POST requests. The body that the request will receive is a JSON document with an attribute for the tree of blocks that represent the assessment and an attribute that contains the JSON-encoded Scratch project.

The plugin is expected to return an HTTP status code of 200 and a JSON response body with the assessment result like this:

```
{
        “Description”: “…”,
        “Pass”: true,
        “Actions”: [
                {
                        “Type”: “…”,
                        “Command”: “…”
                }
        ]
}
```

By implementing a plugin system that relies on communication over HTTP, plugin authors are free to implement their plugins in whichever language they feel most comfortable with. This should encourage extension of the Formative SAGE Assessment project in future iterations.

5.2 Visual Assessment Language

The Visual Assessment Language was implemented using Google’s Blockly library for building visual programming editors (Google, 2016). This section will describe the design of the language and the implementation of the editor that can be used to write assessments using the language.

5.2.1 Language Reference

The Visual Assessment Language grammar is provided below. Terminals in this grammar are represented using **bold** font and described with more detail in the following section. The ↓ symbol represents a vertical connection between two blocks. The → symbol represents a horizontal connection between two blocks. Productions that have the suffix *opt* are optional. As assessment begins with the “assessment” non-terminal.

assessment:
        expectation ↓ trigger-list$_{opt}$

expectation:
        expect → actual → assert → matcher

actual:
        actl-sprite

actl-block
actl-block-type
actl-key-pressed
actl-sprite-touch-sprite
actl-sprite-touch-color

assert:
asrt-should
asrt-should-not

matcher:
mtch-be-present
mtch-be-on-x-y
mtch-point-direction
mtch-move-steps
mtch-say

trigger-list:
trigger
trigger-list ↓ trigger

trigger:
condition action

condition:
cnd-if-pass
cnd-if-fail

action:
actn-say
actn-include-block
actn-exclude-block

5.2.2 TERMINALS

Every terminal in the grammar corresponds to a block that is describe below.

expect

The Expect blog signifies the beginning of an assessment. It connects horizontally to an actual block and connects vertically to a trigger block.

actl-sprite

The Actual Sprite block is used to reference a sprite. It takes a single string argument for the ID of the sprite that is being referenced. It connects horizontally to an assert block.

actl-block

The Actual Block block is used to reference a block. It takes a single string argument for the ID of the block that is being referenced. It connects horizontally to an assert block.

actl-block-type

The Actual Block Type block is used to reference a block type. It takes a single string argument for the type of block that is being references. It connects horizontally to an assert block.

actl-key-pressed

The Actual Key Pressed block is used to express a mapping between a key on a keyboard being pressed and a sprite. It takes two arguments. The first argument must be selected from a drop down menu that is pre-populated with single keyboard characters. The second argument is of type string for the ID of the sprite that is being referenced. It connects horizontally to an assert block.

actl-sprite-touch-sprite

The Actual Sprite Touching Sprite block is used to express a mapping between two sprites. It takes two arguments. Each argument is of type string for the ID of each of the sprites that are being referenced. It connects horizontally to an assert block.

actl-sprite-touch-color

The Actual Sprite Touching Color block is used to express a mapping between a sprite and a color. It takes two arguments. The first argument is of type string for the ID of the sprite being

referenced. The second argument is from a drop down menu that is pre-populated with a list of colors. It connects horizontally to an assert block.

asrt-should

The Should block is used to express an assessment where an action should occur. It connects horizontally to a matcher block.

asrt-should-not

The Should Not block is used to express an assessment where an action should not occur. It connects horizontally to a matcher block.

mtch-be-present

The Be Present block is used to express an assessment that verifies a sprite, block, or block type is present.

mtch-be-on-x-y

The Be on X and Y block is used to express an assessment that verifies a sprite is on specific coordinate in the Scratch stage. It takes two arguments. Each argument is of the integer data type and represents the X or Y coordinate.

mtch-point-direction

The Point in Direction block is used to express an assessment that verifies a sprite is pointing in a specific direction. It takes one argument. The argument must be chosen from a drop down menu that is pre-populated with the following options: right, left, down, up.

mtch-move-steps

The Move Steps block is used to express an assessment that verifies a sprite moves a certain number of steps. It takes one argument. The argument is of type integer and represent the number of steps.

mtch-say

The Say block is used to express an assessment that verifies the sprite says, or prints a message to the screen. It takes one argument. The argument is of type string and represents the message.

cnd-if-pass

The block If Pass is used to express a trigger that is invoked if the assessment passes. It connects vertically to an expect or another trigger block. It connects horizontally to an action block.

cnd-if-fail

The block If Fail is used to express a trigger that is invoked if the assessment fails. It connects vertically to an expect or another trigger block. It connects horizontally to an action block.

actn-say

The Say block is used to say, or print a message on the dashboard if the previous trigger is invoked. It takes one argument. The argument is of data type string and represents the message that should be printed on the dashboard.

actn-include-block

The Include Block block is used to include one block on the Scratch palette builder if the previous trigger is invoked. It takes one argument. The argument is of data type string and represents the ID of the block that should be included.

actn-exclude-blocks

The Exclude Block block is used to exclude one block on the Scratch palette builder if the previous trigger is invoked. It takes one argument. The argument is of data type string and represents the ID of the block that should be excluded.

5.2.2 Editor

The visual assessment language editor is implemented using HTML and JavaScript. It extends the Blockly Code editor example on the Blockly website (Google, 2016) by enabling users to upload assessments to the assessment server. When a user requests that an assessment be uploaded, the editor executes an HTTP POST request against the assessment server with the XML-encoded assessments as the HTTP request body.

5.3 Scratch Editor

The Scratch 2.0 editor is implemented using ActionScript and Flex. The code for the editor was written by the Lifelong Kindergarten Group at the MIT Media Lab and is publically available on GitHub under the GPL v2 license (MIT Lifelong Kindergarten Group, 2016). This Formative SAGE Assessment project extends the fork created by Jeff Bender (Bender, 2015) in two major ways. First, the editor will periodically upload a student's work to the assessment server without disrupting the student. Second, the editor will enable or disable blocks based on the result returned by the assessment server.

5.4 Dashboard

The Dashboard is implemented using HTML and AngularJS (Google, 2016). It leverages the Bootstrap framework (Twitter, May) to display a dashboard with the results of the latest assessment evaluation for an assignment. To access the dashboard, the following route must be used:

/students/{id}/assessments/{id}

This communicates the student ID and assessment ID to the AngularJS controller. The controller will periodically poll the assessment server to retrieve the latest assessment results for the

student and upload the dashboard without requiring a manual refresh of the web page displaying the dashboard.

6.0 Deployment

This section will describe the patterns and technologies used for code version control, integration, and deployment.

6.1 Code Repository

GitHub (Github, 2016) was used as the code repository for the Scratch Editor and Bitbucket (Atlassian, 2016) was used as the code repository for the Dashboard, Visual Assessment Language Editor, and Assessment Server. The table below lists the repositories and their links.

Repository	Link
Scratch Editor	https://github.com/jeffbender/SAGE-scratch/tree/jairo
Dashboard	https://bitbucket.org/sagecoms/sage-frontend
Visual Assessment Language Editor	https://bitbucket.org/sagecoms/sage-editor
Assessment Server	https://bitbucket.org/sagecoms/sage

6.2 Continuous Integration and Deployment

To facilitate current and future deployment of the Formative SAGE Assessment Project, the Jenkins (Jenkins, 2016) open source automation server was used for continuous integration and deployment. A project was created for each of the Github and Bitbucket repositories on Jenkins so that when code is committed to any of the repositories, a job will be started in Jenkins. Figure 11 demonstrates a screen capture of a Jenkins project for the Assessment Server. The job consists of multiple independent steps that are triggered if the previous step completes successfully.

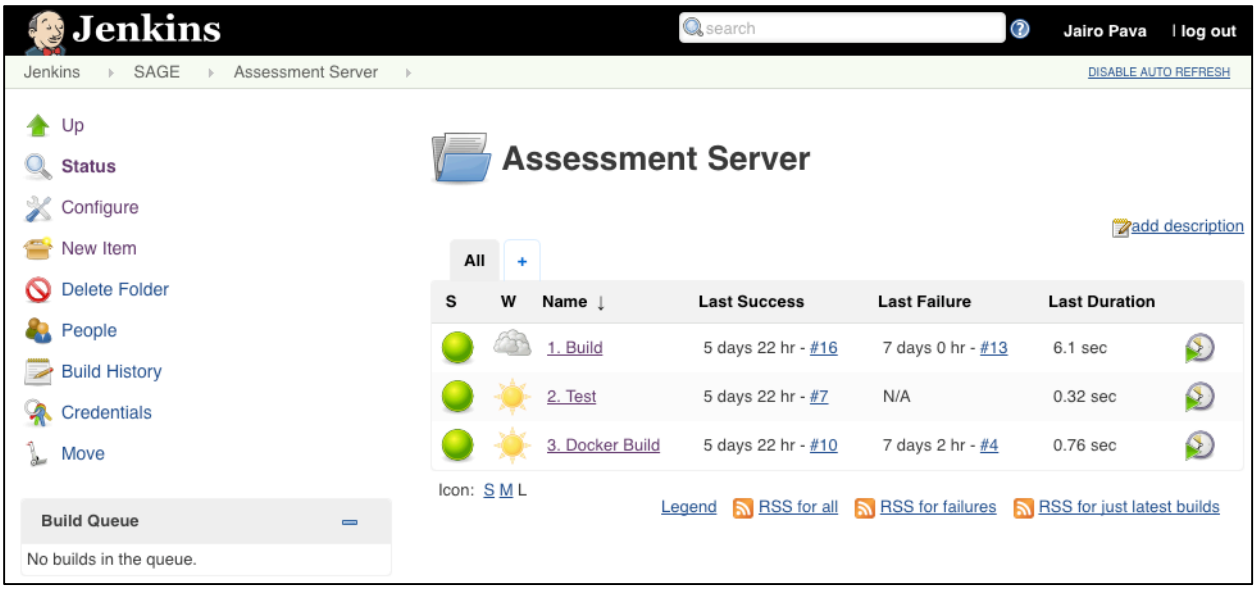

Figure 11 Jenkins

The first step pulls the code from the repository, downloads any required libraries, and compiles the code. If the step succeeds, then the Test step runs. This step executes unit and integration level tests. Finally, the third step builds a Docker (Docker, 2016) image with the compiled executable created in the first step.

Docker allows the image to be packaged into a container with all of its dependencies in a complete filesystem that has everything it needs to run: code, runtime, system tools, and system libraries. This guarantees that it will always run the same, regardless of the environment it is running on. In a separate step, Docker containers running a previous build of the project will be stopped, removed, and replaced with a Docker container running the newest version of the project build. The containers also share a private network that allow them to communicate with each other. This continuous integration and deployment pipeline greatly reduces the amount of time between writing code and shipping to production.

The Jenkins server is hosted on the Microsoft Azure Cloud (Microsoft, 2016) so that when the Docker containers are deployed, the application is publically available on the Internet. The table below provides links to each of the applications in the Formative SAGE Assessment Project.

Application	Link
Jenkins	http://sage-2ik12mb0.cloudapp.net:8080/
Assessment Server	http://sage-2ik12mb0.cloudapp.net:8081/
Visual Assessment Language Editor	http://sage-2ik12mb0.cloudapp.net:8082/app/editor/
Dashboard	http://sage-2ik12mb0.cloudapp.net:8083/index.html#/
Scratch Editor	http://sage-2ik12mb0.cloudapp.net:8084/sage.html

7.0 Future Work

There remains plenty of future work to make the Formative SAGE Assessment Project a good first step towards computational thinking in the classroom. A study should be performed in a classroom environment to evaluate strengths and weaknesses of the project. Students should be asked to use SAGE for a short period of time that will enable an evaluation of the dashboard and visual assessment language components.

The dashboard should be evaluated for its ability to encourage students to continue working on assignments even if they feel stuck. It should also be evaluated for how well it is able to get students to focus on specific computational thinking concepts that the teacher is targeting. The visual assessment language should be evaluated for how easy it is for teachers to use and automate assessments that enable them to focus on subjective evaluation of assignments.

Further work may also be performed on the data that is being captured by the Assessment Server on assignments, assessments, and assessment results. Over time, this data may reveal patterns about how students approach assignments, solve problems, and pass assessments. These patterns may allow the project to automatically generate personalized recommendations instead of relying on pre-determined suggestions that the teacher must provide when creating the assessments. These patterns may also help teachers identify which assessments are actually helping students with their assignments.

Finally, the visual language editor may also be improved to be a better fit in the classroom. An entire web site may be built around the editor that supports multiple teachers, with multiple classrooms and students. The web site can keep track of student progress over time so that teachers can determine which students need special attention to understand specific computational thinking concepts.

8.0 Conclusion

The Formative SAGE Assessment Project was presented in this report. It enables teachers to automate assessments for student SAGE submissions so that they may focus more of their time on subjectively evaluating student assignments. It also enables students to focus on the specific goals of the assignments created by their teachers by providing them with a dashboard that updates in real-time as they work on their assignments. The dashboard provides them with feedback and suggestions that will help them complete their assignments in the manner that their teacher intended.

There is much work left to be done for this project. An evaluation of the project should be conducted in a classroom environment where its strengths may be built upon and its weaknesses addressed. Future work should aim towards further analysis of how the visual assessment language in this project can be improved to better capture a student's computational thinking progress over time.

9.0 REFERENCES

Adobe. (2016, May). *Adobe*. Retrieved May 2016, from Adobe Flash Player: https://get.adobe.com/flashplayer/

Atlassian. (2016, May). Retrieved from BitBucket: https://www.bitbucket.org

Barr, V., & Stephenson, C. (2011). *Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community?* ACM InRoads.

Bender, J. (2015). *Developing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula.*

Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P., & Franklin, D. (2013). Hairball: Lint-inspired Static Analysis of Scratch Projects. *SIGCSE* (pp. 215-220). Denver: ACM.

Brennan, K., Balch, C., & Chung, M. (2015). *Creative Computing.* Harvard University, Graduate School of Education, Cambridge.

Davis, E. A., & Krajcik, J. (2005, April). Designing Educative Curriculum Materials to Promote Teacher Learning. *Educational Researcher*, 3-14.

Docker. (2016, May). Retrieved from Docker: https://www.docker.com

Github. (2016, May). Retrieved from Github: https://www.github.com

Google. (2016, May). Retrieved from The Go Programming Language: https://golang.org/

Google. (2016, May). *AngularJS*. Retrieved from https://angularjs.org/

Google. (2016, May). *Blockly Demos Code Editor*. Retrieved from Blockly: https://blockly-demo.appspot.com/static/demos/code/index.html

Google. (2016, May). *Google Developers*. Retrieved from Blockly: https://developers.google.com/blockly/

Jenkins. (2016, May). Retrieved from Jenkins: https://jenkins.io/

Microsoft. (2016, May). Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/

MIT Lifelong Kindergarten Group. (2016, May). Retrieved from Scratch - Image, Program, Share: https://scratch.mit.edu/

MIT Lifelong Kindergarten Group. (2016, May). *Github*. Retrieved from Scratch 2.0 editor and player: https://github.com/LLK/scratch-flash

MongoDB, Inc. (2016, May). Retrieved from MongoDB: https://www.mongodb.com/

Moreno-Leon, J., Robles, G., & Roman-Gonzalez, M. (2015, September). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *RED-Revista de Educación a Distancia*.

Murphy, C., Kaiser, G., Loveland, K., & Hasan, S. (2009). Retina: helping students and instructors based on observed programming activities. *SIGCSE* (pp. 178-182). Chattanooga: ACM.

Seiter, L., & Foreman, B. (2013). Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. *International Computing Education Research.* San Diego: ACM.

Twitter. (May, 2016). *Bootstrap*. Retrieved from http://getbootstrap.com/