University of Victoria

Department of Software Engineering

SENG 371 Software Evolution, Spring 2019

Simplified Examination Environment

See it works.

Gang of Six

Misha R, Devlyn D, Grant H-M,
Robbie T, Alec C, & Juan C

Table of Contents

List of Figures

1 Introduction

This document discusses the Simplified Examination Environment (SEE) application[1] and codebase with a focus on how it can improve research-focused Python scripts. SEE is targeting researchers who write scripts without vast technical knowledge, yet still need to deploy robust and reliable code while maintaining a primary focus on the their work. This development was formed out of SEE Project 1, the analysis of both the Compute Canada and the One-Machine codebases.

Additionally, this document demonstrates the deployment of an instance of SEE, and how the system can be used to better users' code. SEE operates by allowing users to upload their project files and then pass them through a series of checks, called filters, to test their code. SEE is evaluated on both the effectiveness of a code analysis and how evolvable and scalable the implementation is relative to the prior first iteration of SEE. Lastly, the recommendations for future work of SEE is discussed, including what issues need to be addressed and what features are still to be implemented.

2 Purpose

SEE was developed in response to a case study of a sample Earth Data Store application. The case study followed the evolution of a geospatial data cleaning application from a one-machine system to Compute Canada. The application suffered from a lack of documentation, dependency checks, poor file names, and general bad coding habits, such as including usernames and passwords in plain text. All of these issues made the migration from one-machine to Compute Canada a difficult task, and ultimately led to a further unstable codebase, highlighting the need to reinforce basic coding practices and to use CI tools.

SEE is designed to ensure other codebases are minimizing the effects of software evolution unlike the program in the case study. With the use of SEE, codebases can expect to be more stable, scalable, and evolvable. Used correctly, upscaling from a one-machine system to a cloud computing platform, such as the Evolvable Earth Data Store, would be a much easier experience than that of the case study application.

3 SEE Deployment

In this section, we will discuss the deployment of SEE. First the prerequisites required to set up an instance of SEE and then the setup process of the backend and frontend components.

3.1 Prerequisites

This guide assumes you already have Node.js[2] version 10 or higher, npm or yam, and Git installed on your machine.

3.2 Installation

To deploy SEE, it is recommended to use git to clone the project repository, as shown below, or download the project files from the Github repository[1].

```
~/ > git clone https://github.com/avlec/see.git
```

Then from the project directory install all the necessary tools using npm as shown below, with the project directory assumed to be in the users home folder.

```
~/see/ > npm install
```

Starting the client and server can be done as shown below. Your default browser will automatically be opened.

```
~/see/ > npm start
```

Alternatively, you can start the server by launching the server through the command line with node as shown below. However you will need to manually open your browser and connect to the server, that's why the above is prefered.

```
~/see/ > node server.js
```

3.3 Execution

After starting the server, you can upload files through the client or by using curl and replacing the square file between the square brackets with a file of your choice:

```
curl -X POST localhost:5000/uploadFiles -F "file=@./[examplefile.txt]" -b
cookies -c cookies
```

The `-b` and `-c` flags to curl tell it to maintain a session. Without it, the `see-uploads/` folder where all the file uploads go, will just keep uploading your file as a new user.

To run a filter on the client, click “check script” on the client webpage and it will run all of the filters against your codebase. Alternatively use curl and specify the filter type and file to filter respectively as follows.

```
$> curl localhost:5000/runFileFilter/[runFilterType]/[examplefile.txt] -c
cookies -b cookies
```

You can choose from multiple filter types including checkFileSize, checkLineCount, checkContainsPassword, checkLineLength and runPythonLint.

4 Implementation and Features

SEE is a web based examination tool to take a provided set of Python scripts and review them for linting, formatting, inconsistencies, security vulnerabilities, and other tests. Changes are suggested and recommended to have the users follow better code practices. SEE aims to teach users how to write better, more maintainable code, and leave them with skills and knowledge they can use in all of their projects.

4.1 Features

The application provides users with access to:

- Automatic code linting and formatting
- Codebase security audits
- Warnings for code quality, poor practices, or concerns of security

SEE is implemented as an extendable system that anticipates further future development of features. This is discussed later in section 6, *Recommendations of Future Work*.

4.2 Tools Used

The tools used throughout the project include:

- React - Javascript library for building the web interface
- Travis CI - Continuous integration platform for testing and building SEE
- Github - Hosting and maintaining SEE

Some of the code languages used in this project:

- Javascript: with the Nodejs webserver, React UI, and several filters..
- Python: some of the filters were written in python and called through Javascript.
- HTML & CSS: for the webpages.

5 Project Evaluation

This section will examine how SEE was designed including prototyping design strategies and devops tools used, how SEE could be useful to other organizations, and how evolvable is SEE as a standalone system.

5.1 Prototypes

A mockup of the UI was first developed in Adobe XD and was later expanded upon into a functional prototype. The use of rapid prototypes allowed for the design of the UI to be developed quickly. Additionally, these prototypes didn't require writing any code which would ultimately suffer the effects of software evolution.

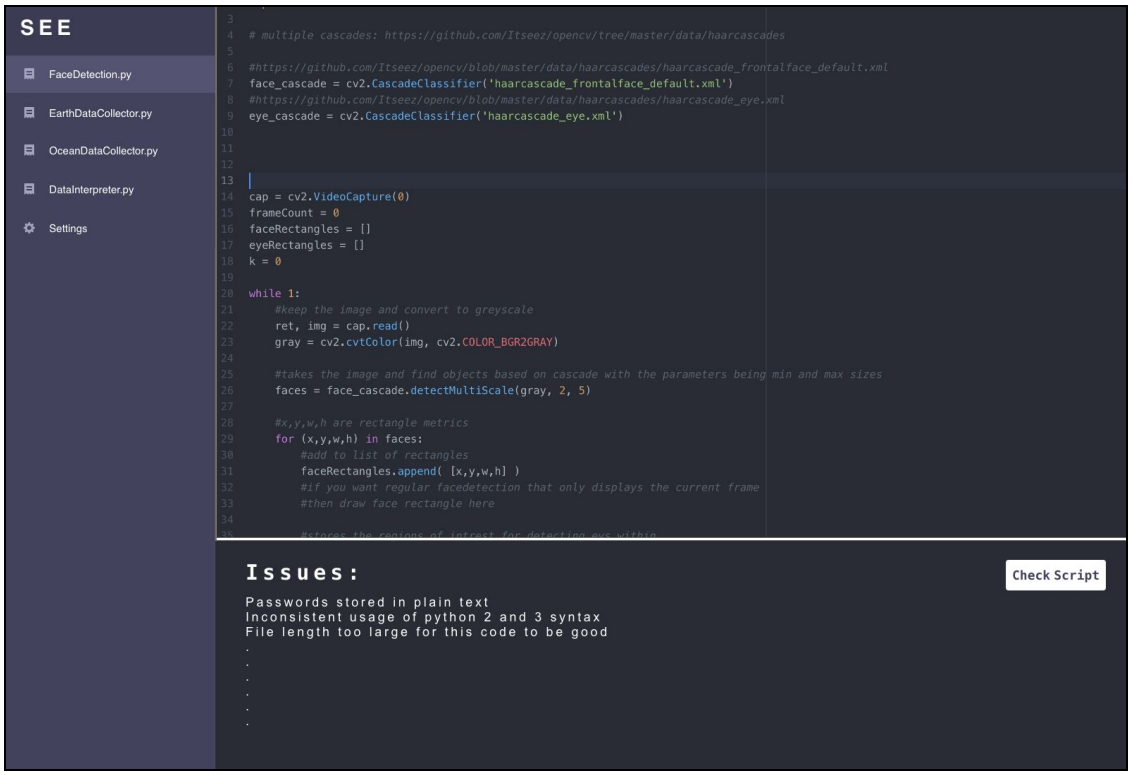

Figure 1. SEE UI Prototype

These prototypes also act as a form of documentation that improved the understanding of the design of the project. Features were proposed and designed with a prototype from which code was then written to match. Going forward, new features can be prototyped, designed, and developed to maintain this evolvability.

5.2 DevOps

While developing SEE, to ensure it did not suffer from the negatives effects of software evolution, we made use of Git version control and the CI/CD tool, Travis CI. Travis CI provides us with information about the quality of the code by running tests and reporting the results on both our Travis CI[3] site and on Github. By having every commit tested by TravisCI and our predefined tests, we can be more certain that we are producing an evolvable and high quality product.

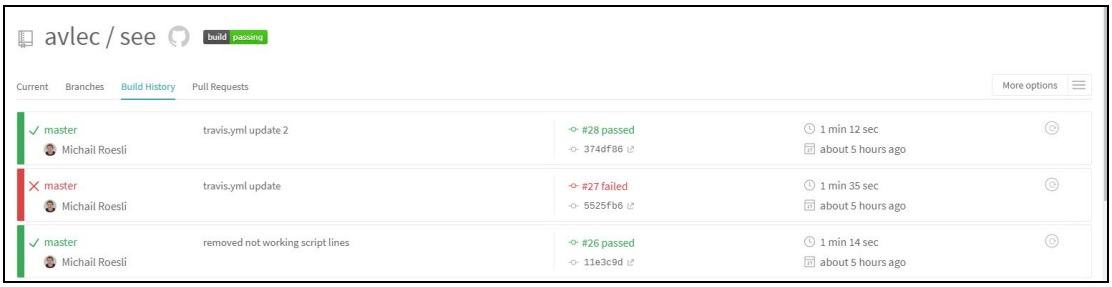

Figure 2. Travis CI Build History snippet for SEE

5.3 Using SEE for Maintenance and Evolution

SEE provides organizations with the ability to audit their software for mistakes commonly made by human developers. Our platform covers the parts of code that most developers wouldn't care about and provides them with the feedback on the changes they're making to their codebases. If One Machine or Compute Canada had used our system, it would reduce the likelihood that software architecture and development antipatterns arise such as spaghetti code, swiss army knife, stovepiping, and copy-and-paste programming.

5.4 Evolvability and Scalability of SEE

The modular structure to the different filters that SEE uses when parsing code keeps SEE evolvable and scalable. With the use of Continuous Integration tools like Travis CI each iteration of SEE, as demand for different and more complex filters grows, will maintain functioning code and a stable codebase.

5.4.1 One Machine

The codebase provided in the one machine was a quick effort to get something working on a smaller scale, thus CI was definitely not a priority for them. However, to ensure that the code being developed for the multi-node compute canada infrastructure was being written properly - which it wasn't - developers would need to test small changes made to the codebase and infrastructure. Moreover, the codebase lacked any kind of effort as the system was simply stovepiped into the compute canada system. They we're missing the steps of reverse

engineering, and reengineering for the new architecture. As this haphazard band aid oriented solution proved to be corrosive with their codebases quality.

5.4.2 Lock-in

One of the primary benefits of using SEE over other code analytics software for the Earth Data Store is that there is no lock-in. SEE functions without tying users into proprietary software and has the ability to drag and drop any coding project new or currently existing.

6 Recommendations of Future Work

- **Multiple file support** and providing a worktree for viewing the currently uploaded files. This is needed for many of the other future work items below as programs like code auditing and analysis operate on entire projects and not single disconnected files.

- **Execution environment detection** via dependency analysis and keywords. For example, using the `aws-cli` package is a strong indicator that the application is running in Lambda or a similar AWS environment.

- **Library detection** by reading `import` statements from uploaded code, SEE could provide a list of used dependencies within the project and encourage users to solidify their external dependencies for dependency pinning[4].

- **Security audits** via Pylama[5].

- **Consistent use of a Python version** automatically by looking at syntax.

- **Optimization**: File/database caching optimizations, speed hacks.

- **Asking questions** for common pitfalls and considerations of a project. Examples:

    - If no `requirements.txt` or similar package management lock file, ask: "*Does your project use any dependencies?"* If yes, recommend they use package manager that uses lock files and dependency pinning [4] such as `pipenv`

    - If there's no license file, ask: *"Is your project licensed? There's no apparent license file"* Then recommended a basic template for the license file such as MIT or GPL.[6]

    - If there's nothing similar to `readme.md` or `docs/`, ask: *"Do you have any documentation to explain the project, how to use it, and reproduce execution?"*

- **Interconnectivity** analysis of uploaded source files and determining how they interact. This would allow us to show codebase complexity and help the user modularize code. Note that this is a very hard problem even for humans to do.

- **Recognize design patterns** and anti-patterns in code.
- **Additional automated tests** for any filters implemented. Having automated tests ensure that even after changing a filter it still functions properly.

7 Glossary

This section of the document explains and defines abbreviations and jargon used throughout the document.

Term	**Description**
Anti Pattern	A high level description of common defective implementation and processes within organizations.
CD	Continuous deployment, referring to the frequent deployment of changes made through CI.
CI	Continuous integration, referring to the continuous changes made to a software project, of which only the good are integrated back into the software project.
Canada Compute Cloud / Compute Canada	This is a cloud infrastructure provided by the Canadian government for usage by Canadian researchers and scientists to execute performance demanding tasks
Distributed Cloud Computing	Cloud infrastructure that is physically distributed and duplicated in multiple areas of the world for redundancy and to allow a cloud application to move as needed. Often abstracted to appear as a single server and service.
One Machine	Referring to the usage of a single machine to perform some computational task
The Platform/System	The system being described by this document

8 References

[1] SEE GitHub repository: "avlec/see", GitHub, 2019. [Online]. Available: https://github.com/avlec/see. [Accessed: 08-Apr-2019].

[2] Node.js Foundation, “Node.js,” *Node.js*. [Online]. Available: https://nodejs.org/en/. [Accessed: 08-Apr-2019].

[3] “avlec/see - Travis CI.” [Online]. Available: https://travis-ci.com/avlec/see. [Accessed: 10-Apr-2019].

[4] US Government, "Pinning dependencies for production" [Online]. Available: https://before-you-ship.18f.gov/infrastructure/pinning-dependencies/ [Accessed: 08-Apr-2019]

[5] Pylama, "klen/pylama", *Pylama: Code Audit Tool*, GitHub 2019. [Online]. Available: https://github.com/klen/pylama [Accessed: 08-Apr-2019]

[6] Choose a license, [Online]. Available: https://choosealicense.com/licenses/ [Accessed: 08-Apr-2019]