Code Inspection Report

Course Evaluation System

Team EVAL

Jovon Craig, Sam Elliott, Yuanqi Guo, Robert Judkins, and Stanley Small

Client: Dr. Harlan Onsrud

March 14, 2019

University of Maine - Spring 2019 - COS 497
Instructor: Professor Terry Yoo

Course Evaluation System

Code Inspection Report

Contents

1 Introduction

Team EVAL is creating a system to more efficiently create and distribute post-semester teaching evaluations. Our client wants us to build such a system because the current one used by the University of Maine is too difficult to use. To ensure that we are on the right direction, our system must be properly inspected to be free from defects and meet all the functional requirements as laid out in the System Requirements Specification.

1.1 Purpose of This Document

The code inspection report is a detailed overview of the walkthrough our team made to ensure that the course evaluation system works as intended. The first section of the document mentions the coding conventions that the team used as guidelines, as well as a list of defects we checked during the review. The second section details our inspection process, how it deviated from a typical inspection, and Team EVAL's reflections. The last sections describe the system components inspected and the defects we found.

This document is intended for the development team, the product client, Dr. Harlan Onsrud, and potential users of the system. Team EVAL needs this document to fix the defects that we found during inspection and keep the defect checklist in mind. Dr. Onsrud needs it to confirm that Team EVAL is meeting requirements and caring about code quality. Lastly, the document helps the software's users by telling how the actual modules differ from how they were designed in the SDD.

1.2 References

Craig, J., Elliott, S., Judkins, R., & Small, S. 29 October 2018. *System Requirements Specification.*

Craig, J., Elliott, S., Judkins, R., & Small, S. 16 November 2018. *System Design Document.*

Van Rossum, G., Warsaw, B., & Coghlan, N. (2001, July 5). *PEP 8 – Style Guide for Python Code.* Retrieved from https://www.python.org/dev/peps/pep-0008/

Goodger, D., & Van Rossum, G. (2001, May 29). *PEP 257 – Docstring Conventions.* Retrieved from https://www.python.org/dev/peps/pep-0257/

Google JavaScript Style Guide. (n.d.). Retrieved March 4, 2019, from https://google.github.io/styleguide/jsguide.html

1.3 Coding and Commenting Conventions

Team EVAL has adopted several guidelines for proper code and commenting to have a more consistent code base. In code inspections these standards will be referenced better understand what kinds of defects may be present in the code inspected.

For the Python code in the back-end API, the team is following the official Python style guide, PEP 8, and the docstring conventions in PEP 257. Available on the Python website, PEP 8 and 257 were written in 2001 by the creator of Python, Guido van Rossum, and other core developers. Both guides in turn were derived by van Rossum's original style guide for Python. PEP 8 covers topics such as indentation, line length, imports, whitespace, comments, and naming. PEP 257 has brief guidelines on documentation strings.

For the JavaScript code written on the front end, Team EVAL is following the Google JavaScript Style Guide. Google uses JavaScript so much for its front-end projects that it wrote conventions on how to properly write in the language. The standards in this guide are followed by Google's developers for their own development. The style guide includes guidelines on issues like source file structure, indentation, whitespace, comments, literals, naming, and JavaScript's documentation language, JSDoc.

1.4 Defect Checklist

The team created a list of possible defects that could be encountered during our inspection. Much of the defect list references the style guides mentioned in the previous section. Each defect falls under one of four

categories: violations of coding conventions, control flow errors, flaws in the system's security, and violations of commenting standards. The list is shown in Table 1 on the next page:

Table 1: Defect checklist

Category	**Statement**
Coding Style	Indentation is done with tabs instead of spaces
Coding Style	Indentation amount hurts code clarity
Coding Style	Multiple imports on the same line
Coding Style	Extraneous whitespace around punctuation
Coding Style	Unnecessary blank lines
Coding Style	Unnecessary compound statement
Coding Style	Too many units in a single script file
Coding Style	Return types in same function are inconsistent
Coding Style	Double quotes in string literal (JavaScript)
Coding Style	No braces in control structure (JavaScript)
Coding Style	JavaScript name is not in lower camel case
Coding Style	Python name is not in lowercase with underscores
Coding Style	Constant name is not in all-caps
Commenting Style	Comment is unnecessary for understanding
Commenting Style	Function or class has no docstring
Commenting Style	Function docstring does not mention parameter and return types
Commenting Style	Multi-line docstring has no summary line
Control Flow	Off-by-one error in a loop
Control Flow	Non-explicit exception handling
Control Flow	Too many paths in a control structure
Security	Default password used for authentication
Security	No authentication in code requiring security
Security	Unsanitized user input
Security	Open TCP/IP ports
Other	Function has no unit test(s)

2 Code Inspection Process

The code inspection was conducted with all members of Team EVAL present in a single sitting. The code was reviewed beforehand by each member, and a preliminary check was completed by each author. The following sections document the findings of our inspection.

2.1 Description

The code inspection process that Team EVAL followed was more informal than a typical Fagan inspection. Our team compared the process that we followed at the inspection meeting to a walkthrough as described by Williams in the text Intro to Software Engineering. Our process diverged from the typical inspection in an attempt to save time, allow for a more flexible meeting style, and a more casual atmosphere.

Before the inspection, the team decided the time and location of the meeting and who would take each role. Next, Stan and Jovon wrote the defect checklist so that the team could use it as a reference for what defects could be found. During the inspection, Stan led the proceedings as the "moderator", and Sam was the "recorder", typing down any defects encountered. The author of a module took on the "author" and "reader" roles. All members of the team were "inspectors", pointing out flaws in the code. Some flaws referenced the style guides described earlier in this report.

2.2 Impressions of the Process

Our inspection process proved to be very effective. We were able to find a number of defects or simple fixes that may not have been caught without the inspection. It also helped the team focus on continuity and consistency in our code. It was very beneficial to sit down as a team and walk through every file in our growing project. As the project grows in size, it is easy for defects to fall through the cracks when trying to produce high-quality code. Sitting down as a team and systematically going through every file in our repository caused us to double-check work that we previously may not have reviewed. Since our project has several separate components, it was useful for each of us to see files that we may have written and to have the original authors explain them.

We were able to find defects and discuss possible solutions to them quickly. We did not waste time delving into exact solutions of larger issues, but many of our defects were quick fixes and it was important to discuss them as a team. The process seemed beneficial enough for a team and project of this size that it would be useful to do the exact same process several times throughout development.

The best modules as determined by the team were contained in the back-end files. While some of these files were generated by tools and frameworks, the code clearly followed the coding standards outlined in this document and contained more than sufficient documentation. Specific files which showed great promise include teachers_controller.py and test_teachers_controller.py.

Many of the worst modules, specifically QuestionsForm.js and EnrollForm.js, were contained in the front end made with the React framework. The code showed a distinct lack of understanding of the framework and deviated quite strongly from the system design document. As a consequence, the team has decided to revisit these sections of the code to better reflect decisions made early in the design process.

2.3 Inspection Meetings

We held one inspection meeting to make our code inspection report. This meeting took place in a physical location, Boardman Hall 136. It occurred on March 6, 2019, from 3 PM to 4:15 PM. Everyone on Team EVAL (Jovon, Sam, Tom, Robert, and Stan) participated in the inspection. Stan was the moderator of the inspection, Sam was the recorder, whoever wrote a certain modular unit served as the author and reader, and all team members were inspectors. The meeting covered every code unit in the system that is not automatically generated, including testing files, JavaScript files, and Python scripts.

3 Modules Inspected

In our inspection meeting, we looked through any module that was integral to our course evaluation system. The team defines a module as any individual script that the system can run. Table 2 on the next page lists each module examined, its functionality, and how it fits into the design mentioned in the SDD. Note that a few modules have been replaced by new ones since our inspection.

Table 2: List of modules inspected

Name	Functionality	Relation to System Design
.travis.yml	Defines which tests are used in continuous integration	For testing, not in system design document
docker-compose.yml	Defines the docker-compose command	For project building, not in system design document
About.js	What is displayed on the About page	Part of React front end
App.js	Old file which ran the application, replaced by AppRouter.js	Part of React front end
AppRouter.js	Routes the application to different pages, contains the current state of the app as a whole	Part of React front end
CourseForm.js	Page on which users define a new evaluation form	Part of React front end
EditCourse.js	Page on which users edit old courses, replaced by CourseForm.js	Part of React front end
EnrollForm.js	Page on which users define who takes the evaluation form	Part of React front end
FAQ.js	What is displayed on the FAQ page	Part of React front end
Home.js	What is displayed on the Home page	Part of React front end
Landing.js	What is displayed on the Landing page	Part of React front end
Login.js	What is displayed on the Login page, allows users to log in	Part of React front end
QuestionForm.js	Page on which users enter which questions they want in the survey	Part of React front end
Index.js	Used in construction of JS app	Part of React front end
insert_mock_data.sql	Inserts mock data into the database	For testing, not in system design document
test_teachers_controller.py	Runs tests for back-end API	For testing, not in system design document
teachers_controller.py	Processes API calls	Part of Python API
limesurvey.py	Interfaces with LimeSurvey	Part of Python back end

Some of the modules in the system are not yet completed. User login has not yet been completed, but will be finished after spring break. Additionally, some pages originally promised to the client have not been implemented. The team is focused on the most essential elements of the application, and has notified the client accordingly.

There are several differences between the actual design of a few of the modules and what was proposed in the System Design Document. The front end is supposed to exploit the features of the React library for user interaction, but Sam and Robert instead used mostly HTML for this. They are currently reworking the front end so that React.js is used properly.

In the back end, the endpoint "GET publish" was added so that the API could create a .txt file of a survey and publish it on LimeSurvey. The "POST results" endpoint was removed because there seemed to be no need for it. Also, the API now retrieves a JSON object of survey results instead of a .txt file. The team did not mention using an external library ("limesurvey.py") to interface with LimeSurvey because we did not know it would be helpful.

4 Defects

During the inspection, the team found multiple defects in the system, spread across many different components. Table 3 on the next page gives a list of all the defects that we found. Each defects falls under one of five categories: correctness to the functional requirements, coding conventions, commenting conventions, user friendliness, security, and miscellaneous flaws.

Table 3: List of defects found

Module	**Category**	**Description**
.travis.yml	Other	Lacks back-end API tests
.travis.yml	Commenting	Does not have enough comments
docker-compose.yml	Commenting	Lacks comments
Front-end modules	Commenting	No header comment in each file
Front-end modules	Commenting	No docstrings
Front-end modules	Coding Convention	Double quotes used in some strings
Front-end modules	Correctness	No input validation
Front-end modules	User Friendliness	No error messages
Front-end modules	Security	Log-in validation ignored after authentication token generated
Front-end modules	Other	Files are not in subdirectories
Front-end modules	Other	No tests
Front-end modules	User Friendliness	CSS buttons too big
About.js	Commenting	No comments
About.js	User Friendliness	Displayed text is outdated
Form.js	Coding Convention	Some names not in lower camel case
Form.js	Correctness	No global variable for API
EditCourse.js	Other	Redundant file
EnrollForm.js	Coding Convention	Has too little whitespace
QuestionForms.js	Coding Convention	Poor formatting
QuestionForms.js	Other	Too long to be one script
Results.js	Correctness	Displays mock data
Back-end modules	Other	Has unnecessary “admins” scripts
insert_mock_data.sql	Correctness	Does not insert all required survey tags
test_teachers_controller.py	Commenting	Lacks comments
teachers_controller.py	Coding Convention	Has multiple imports in one line
teachers_controller.py	Security	Uses default MySQL username and password
teachers_controller.py	Security	MySQL password exposed in repo
teachers_controller.py	Commenting	Too few comments
teachers_controller.py	Coding Convention	Some lines are too long
teachers_controller.py	Coding Convention	Inconsistent string formatting methods
limesurvey.py	Commenting	Lacks comments

A Agreement Between Customer and Contractor

This page shows that all members of Team EVAL and the client, Harlan Onsrud, have agreed on all the information in the code inspection report. By signing this document, Team EVAL and Dr. Onsrud approve the coding conventions used, the information about the inspection process, and the descriptions of the defects and inspected modules.

The team will follow a process in the case that the document is changed after we sign it. First, the team will write a rough draft of the changes to be made to the document. Second, all team members and Harlan Onsrud will sign the document agreeing to the changes. Finally, the team will make the changes to the final copy of the document.

Name	***Signature***	***Date***
Jovon Craig	_____	_____
Sam Elliott	_____	_____
Yuanqi Guo	_____	_____
Robert Judkins	_____	_____
Stanley Small	_____	_____
Dr. Harlan Onsrud	_____	_____

Customer Comments: _____

_____

B Team Review Sign-off

This page shows that all members of Team EVAL have reviewed the code inspection report and agreed on its content. By signing this document, the team members agree that all information about Team EVAL's code inspection is accurate, and there is nothing in the document that is a source of contention.

Name	***Signature***	***Date***
Jovon Craig	_____	_____
Comments:	_____	
_____		
Sam Elliott	_____	_____
Comments:	_____	
_____		
Yuanqi Guo	_____	_____
Comments:	_____	
_____		
Robert Judkins	_____	_____
Comments:	_____	
_____		
Stanley Small	_____	_____
Comments:	_____	
_____		

C Document Contributions

Stanley Small discussed the status of various modules and helped to describe the outcomes of the inspection. He proofread the document for final submission. Stan contributed approximately 20 percent of the document.

Jovon Craig wrote the purpose of the document, references, coding conventions sections, part of the defect checklist, description of the inspection process, and list of defects. He also made several revisions to the document. Jovon contributed about 40 percent of the document.

Yuanqi Guo wrote more content for the introduction and revised the formatting in the document. Tom contributed about 5 percent of the document.

Sam Elliott added the table of the modules that the team inspected and made several small additions to the document. Sam contributed about 20 percent of the document.

Robert Judkins wrote the section stating the team's impressions of the inspection process. He contributed about 15 percent of the document.