ASRC Onboard Online Learning Resource

Validation Plan

Prepared by: Christopher Pham

Team Parrots

Date: 11/27/19

Table of Contents

1. Introduction

The purpose of this document is to organize the ASRC Onboard Online Learning Resource project into manageable sections for a software development view, and define all parameters to prepare for testing the system.

These parameters may include, but are not limited to:

1.) Global definition of scope for testing
2.) Global assumptions that apply to all project sections
    a.) System assumptions
    b.) Qualifications for people who perform testing
3.) An overall definition of risk that determines the scale of risk measurement
4.) Individual risk analysis for each section
5.) A risk-based decision for each section to determine how rigorous testing shall be

Defining all the necessary parameters ensures that all requirements will situationally be tested thoroughly based on a risk analysis. In conjunction with the highly detailed Test Plan and Test Scripts, this Validation Plan demonstrates that the ASRC Onboarding Online Learning Resource does indeed meet all defined requirements.

2. Scope

The overall scope of this validation encompasses the whole of the Onboarding Learning Resource project as a standalone web service. The validation only needs to cover the project requirements, as all code was developed without dependencies from external resources or previous code written for this project.

The validation will be limited to most of the requirements from the requirements document. Some requirements may be combined with others to make sense in the context of testing. The requirements will also be organized differently than in the requirements document; this organization will be covered in section 4: Validation Organization/Overview.

3. Testing Approach

The overall methods for testing the Onboard Learning Resource system will include ad hoc testing, unit testing, integration testing, and system testing. Because the system is based around a web server, no automated testing scripts will need to be run and will be more trouble than it is worth to create.

Unit testing will be used for tests that check if individual scripts are functioning properly, etc.

Integration testing will be used to test functionalities like checking passed data from webpage to webpage, passing data from webpages to the mySQL database, etc.; these aspects need more than a single section of the project to successfully test.

System testing will be used for tests that confirm the validity of global session variables, sitewide user permissions, etc.

In general, the role of ad hoc testing is to: shallowly check if pages display in the correct format, if pages display the correct information, if page redirects work as intended, if global session variables are set correctly, etc. Essentially, if some informal test is thoroughly repeated during development, it will lessen the need for in depth testing in that area. Ad hoc testing has not been documented in the project.

4. Validation Organization/Overview

This overview outlines the general sections that the document will be analyzing, particularly in section 7. Validation Based on Risk.

Nonfunctional requirements

A. Testing web server on hosting system (a Unix server)
B. Testing the web application on the Google Chrome browser

Functional requirements

Front end

1. Testing each page display
2. Testing if all CSS works

Back end

1. User management
    a. Testing the login system
    b. Verifying valid privilege levels
    c. Verifying password hashing
    d. Session verification
    e. Testing account creation and deletion
2. Quiz management
    a. Testing different quiz **creation/editing** parameters
    b. Testing **deletion** of a quiz
    c. Testing quiz visibility (public/private to other users)
3. Quiz taking
    a. Testing question type displays
    b. Testing grade calculation/metric calculation
4. Database procedures

5. Assumptions

Implementation Assumptions

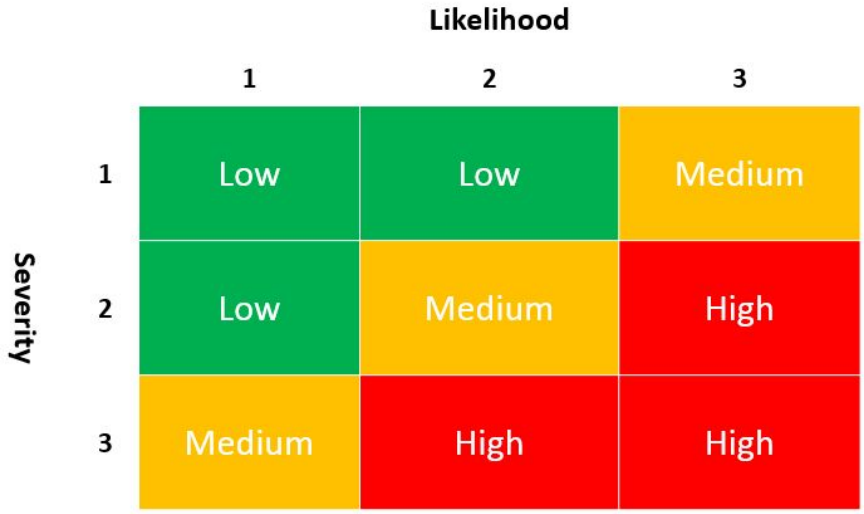
1. The implemented Onboard Learning Resource system will be hosted on a private server from ASRC that is able to easily run a small local website.
2. The server will have the correct versions of Apache and MySQL database installed.
3. The server will be backed up before any changes are made to its code.
4. The database will be backed up periodically in the case of any bugs that incorrectly change any data.
5. The system manager that manages the server (or anyone testing the system) should be comfortable with general web development (HTML and CSS) and know his/her way around the backend of a server.

6. Risk Definition and Validation Depth

Due to the nature of the project itself, the development of the Onboard Learning Resource system generally has a very low overall code risk. There are no hardware parts that may fail and possible faulty grading or faulty quiz taking systems will have both low likelihood and severity. Thus, system testing will suffice for a large bulk of the listed bugs.

The only high risk aspect of the system that exists revolves around the security of the website due to the context of where the system will be implemented in the future; it will be integrated into a server hosted by ASRC Federal, possibly on the open internet or possibly on a secure private network. If the system is openly accessible on the internet, user security will probably be the number one risk to look out for.

We will be using this particular 3x3 risk diagram to categorize the risks of individual system parts:



For severity reference, level 1 severity will refer to a feature that functions unexpectedly, but is still 100% usable; for example, the ASRC logo not being properly aligned on the homepage is unexpected, but doesn't affect the overall usability of the feature. Level 2 severity refers to situations where the bug only affects the functionality for that specific system part or at least a very small amount of the system; an example would be an inability to delete a quiz from the database. This would ruin the functionality of the quiz deletion system but not affect any other system features. Finally, level 3 severity means that if something goes wrong, it will stop a large chunk to the whole system from functioning. This also refers to issues that may cause irreversible changes, like overwriting important information to the database.

For likelihood reference, a level 1 likelihood will classify problems that will almost never arise. Chances are slim to none for a problem to occur during the system lifetime for a level 1 likelihood, even when updating the codebase. Level 2 likelihood will classify issues that have a small chance to appear with normal system use, and a slightly higher chance to appear when changing code throughout the system. Level 3 likelihood will classify problems that have a moderately chance to appear with normal system use, and have a very high chance to appear when changing something in the code base. Likelihood levels generally rise due to an increase in code complexity, whether it has to do with languages interacting with each other, or multiple calls to the complex database.

Low risk classifications will generally need little to no testing, as ad hoc testing during development will usually be enough. If, however, low risk situationally needs testing, system testing will be implemented. Medium risk classifications will need slightly more in depth testing, usually resulting in integration testing or system testing. High risk testing will employ in depth testing, and will most likely include unit testing at the very minimum; implementation testing should generally be used as well.

7. Validation Based on Risk

This section aims to divide the system up into manageable parts of things that could go wrong, and situationally define how in depth testing should be taken. As a whole, all of these system parts should set up testing parameters for all of the requirements in the project Trello Board at the current time of writing this Validation Plan.

Nonfunctional Requirements Testing

A. Host server going offline:

Running the actual system requires a properly functioning host server, with all the code hosted on a Unix server, running Apache 2.4 and MySQL. The webserver can go down due to factors like physical conditions (electricity going out), a very high number of users on the website, or physical webserver maintenance.

a. Severity: 3 - Server downtime will not allow anybody to access the system.
b. Likelihood: 1 - Physical conditions like electricity going out at the place of the server host will likely almost never occur. Not many users should be accessing the system since

functionality is only limited to ASRC employees, and server maintenance is usually planned.

c. Mitigation: ASRC generally should take very good care of their systems to reduce server downtime. In the case of data loss during server downtime, backups of the database will be available, as based on the assumptions listed in section 5 of this Validation Plan.

The overall risk of problems for this requirement is medium. Considering the extremely low likelihood combined with the very reliable mitigation plan, the risk moves to low, and testing (if any) for the capabilities of the host server should be kept to a minimum.

B. System not able to run on Google Chrome:

For some reason, the system may not allow the user to display html correctly, due to an outdated browser or other reasons.

a. Severity: 1 - The website could bug out in the Chrome browser, resulting in very messy looking HTML with no formatting.
b. Likelihood: 1 - The odds of a browser not being able to display a webpage is extremely low.
c. Mitigation: Google Chrome on ASRC employees' computers should be updated so it is never outdated.

The risk for this possible bug is extremely low, and will be covered by ad hoc testing. The system is not required to run on any other browser than Google Chrome, so applications like Mozilla Firefox and Microsoft Edge will not be tested on.

Functional Requirements Testing

Front end

F1. Individual pages display incorrectly or generate HTML errors

Because of faulty HTML, some pages may not display correctly, if at all.

a. Severity: 2 - Having a faulty page could ruin a small chunk of functionality (for example, not being able to view the "Grades" sections).
b. Likelihood: 1 - HTML is generally not very complex, and ad hoc testing will catch most of these bugs before they occur during implementation.
c. Mitigation: HTML files should be checked by a system manager.

The risk for this bug is low, and will be covered by ad hoc testing. However, if more testing is needed, we may use system testing and individually visit all webpages in a browser to check for this error.

F2. CSS fails to display HTML pages as intended

Some of the global CSS code that applies to all webpages may not function on some pages.

a. Severity: 1 - HTML is still viewable with faulty CSS code.
b. Likelihood: 1 - Linking CSS code to HTML code can easily be done with correct naming conventions.
c. Mitigation: The CSS file and offending HTML file(s) shall be inspected by a system manager. Normal use of the system shall provide further informal inspection.

The risk for this bug is extremely low, and will be covered by ad hoc testing. However, if there is an absolute need for formal testing, then system testing should be used to check if every page has aesthetic CSS implemented.

Back end

B1. User management

a. Dysfunctional login system

User input may not be passed correctly from HTML code to the database, or login validation may incorrectly deny a user access to the Onboarding system.

a. Severity: 3 - Not being able to log in means not being able to access the whole website.
b. Likelihood: 1 - These bugs will be caught in ad hoc testing, as developers will constantly test logging in during everyday development.
c. Mitigation: Correctly passing data from a webpage will be checked on the HTML/PHP side, and login information and login validity functionality will be checked on the database side if users are unable to log in.

The risk for this bug is medium since severity levels are high and likelihood levels are low. Integration testing between the HTML/PHP webpage itself and the database will be used to ensure information is correctly passed, that users are able to log in to the system, and that faulty login information will not allow a login. System testing should also be used to see possible errors from an actual user perspective.

b. Invalid privilege levels between employees and mentors

Regular employees may be able to access pages that only mentors are supposed to access, due to incorrect session permissions.

a. Severity: 3 - Employees with mentor privileges would be able to change almost anything in the database and see correct quiz answers for every quiz. Essnetially, an employee with mentor privileges would be able to do a lot of harm to the database information, intentional or not.
b. Likelihood: 1 - Privilege levels that are stored in the website's session variables were implemented near the beginning of development, and have been reliably tested using ad hoc testing and integration testing with global privilege levels and its respective mentor pages. Implementing these privilege levels is not very complex, and will most likely never return an error.
c. Mitigation: If this error does appear, the Onboard quiz system will be taken down and every page using a privilege level session variable will be inspected to ensure that employees may not access mentor webpages.

The risk for this error to occur is medium. Thus, integration testing between privilege level session variables and mentor webpages will be used. However, system testing will cover the lower level testing as well just by looking at the webpages from different user perspectives (mentor and employees).

c. Incorrect password hashing

Users of the Onboarding system may not have their passwords correctly hashed, which raises a security issue if user accounts get hacked.

a. Severity: 2 - Having unhashed passwords is inherently not secure, but an assumption found in Section 5 of this validation plan assumes that the webserver will be private only to ASRC employees. This may be dangerous if this assumption is not true.
b. Likelihood: 1 - Hashing passwords is not complex, and is very easy to implement. The lack of complexity makes the code much more reliable.
c. Mitigation: The only mitigation to a user getting hacked from incorrect password hashing is to change the hacked user's password and make sure the hashing mechanic works properly by going into the backend of the database and checking everything.

The risk for incorrect password hashing to occur is low based on medium severity and extremely low likelihood. The only testing needed for this is unit testing, just to check if a hashed password is stored in the database rather than a plain text password.

d. Faulty session verification [system testing]

Faulty verification of session variables may cause a user to stay logged in after logging out. They also might stay logged in for extended periods of time, which is not desired; too many users on the system could needlessly use server power.

a. Severity: 2 - In the context of system implementation, there is not much danger to have a few ASRC employees constantly logged into the Onboarding system. However, not being able to log out without manually clearing browser cookies brings up a security issue if computers are shared at the workplace.
b. Likelihood: 1 - During development, setting and unsetting session variables is relatively simple. As the code is not complex, the likelihood for faulty system verification is very low.
c. Mitigation: If the error does appear, system managers will be able to check the HTML code used in the logout functionality to ensure that all session variables are being reset.

As this bug overall has a low risk, there is not much need for testing. Basic system testing to check the boundary cases of logging out and checking overall session verification is enough to prevent this bug.

e. Faulty account creation and deletion [integration and system testing] (check front end and database) (check database procedures as well)

Accounts could fail to be created, or be created with the wrong permissions. Account deletion could fail to delete users and clog up the system with many dead accounts.

a. Severity: 3 - Creating accounts with the wrong permissions could be detrimental to the system; employees could have admin privileges and be able to delete other users from the Onboard Learning system. The failure to create accounts could mean that no more admin accounts could be created, resulting in little to no admins to manage the website. Finally, the failure to delete users would clog up the database, and ruin new accounts that attempt to have the same name as a supposedly "deleted" account.
b. Likelihood: 2 - The individual database procedures that handle user creation and deletion are relatively simple. However, passing the data from an HTML form through to the database procedures is more complicated, resulting in a raise of likelihood.

c. Mitigation: The system would be shut down as soon as this bug is found, and would be inspected to ensure that 1.) the database procedures are working correctly and 2.) the parameter passing from the webpage to the procedure is working correctly. If needed, the system managers may revert back to a previous day's database backup to fix any irreversible changes.

Based on the high severity and medium likelihood of this potential bug, there is a need for in depth testing. The individual database procedures for user creation and deletion should not need to be tested using unit testing, but passing all types of parameters through the HTML form on the website should be tested using integration/system testing.

B2. Quiz management

a. Unpredictable quizzes due to unconventional quiz creation/editing input

Entering all kinds of data (say, for example emojis) into text boxes where they aren't supposed to be, or uploading a non-csv file using the "upload quiz" feature could result in unpredictable results.

a. Severity: 3 - Although results of entering strange parameters to quiz creation/editing may be unpredictable, these results are limited to affect only that particular quiz, as all the quizzes are independent in the database. Uploading different files than .csv files could have extremely unpredictable negative outcomes.
b. Likelihood: 1 - The database is designed to accept all kinds of plain text (including characters like emojis) and a specific limit to what kind of file is allowed to be uploaded isn't very complex to code and thus, the likelihood for these unpredictable bugs to appear is very low.
c. Mitigation: The offending quiz that has faulty data would be removed from the database. If a non-csv file were to change the website in some way, system managers would be able to revert the website or the database back a day, due to one of the assumptions.

This bug has an overall medium risk. Implementation testing should be used to test how many types of strange text is written to the database. System testing should also be used to check if non-csv files are successfully rejected from being uploaded.

b. Unsuccessful deletion of a quiz [integration testing]

When deleting a quiz, the quiz may not be successfully deleted and stay in the database.

a. Severity: 1 - An undeleted quiz will not affect the website's overall functionality and will just sit there without causing problems.
b. Likelihood: 1 - The procedure written for this is very simple, and there is only one parameter to be passed, so there is a low chance that errors will occur.
c. Mitigation: If this bug was stumbled upon, a system manager would check the procedure for removing a quiz and check that the correct parameters were successfully passed

Because of the extremely low risk, this aspect of the system would only need basic integration testing to check. The idea would be to remove a sample quiz from the front end, and check the database to see if the quiz was actually deleted.

c. Unsuccessful implementation of quiz visibility [System testing]

If quiz visibility isn't set up correctly, other users may be able to see and take quizzes that are supposed to be private.

a. Severity: 1 - There is not much consequence if this bug occurs. A user would just be taking a supposedly "private" quiz and receiving a grade as normal. Although this ruins the quiz owner's visibility toggle, this bug does not affect any other part of the system.
b. Likelihood: 1 - A visibility flag on each quiz is easy to implement and toggle on and off. Due to this lack of complexity, this bug will most likely never appear.
c. Mitigation: If this bug were to occur, a system manager would simply access the database and remove the grade for the "private" quiz after fixing the functional visibility toggle.

The risk for this system aspect is extremely low. Thus, only system testing is needed to check if private quizzes from other users show up for non-mentor accounts.

B3. Quiz taking

a. Incorrect format per question type (FR, MC, SATA) **[system and integration testing]**

Question pages need to be formatted based on their question type: free response, multiple choice, or select all that apply. Unformatted layouts may result from buggy HTML logic.

a. Severity: 2 - If the layout for the question types aren't correct, incorrect data might be passed from a user answer. Thus, multiple choices will not display correctly, the user is bound to answer the question incorrectly, etc.
b. Likelihood: 2 - The logic for redirecting to pages that support different question types is very simple. It takes one parameter from the previous page and displays question information directly from the database. However, the procedurally generated HTML code to display all of the question information is more complex, resulting in a higher likelihood for there to be bugs.
c. Mitigation: If this error were to occur, the logic for redirecting to different question type pages and HTML code for the offending question types would be debugged. A system rollback to fix bad grades is not needed since users can just retake failed quizzes after the error is fixed.

The overall risk for this error is medium. System testing should first be used as a quick scan for any bugs. If any are found, then integration testing should be used to check HTML logic.

b. Incorrect quiz grade calculation/general metrics calculation

Grade calculations could be incorrectly computed due to logic errors in HTML code.

a. Severity: 2 - Incorrect grade calculations would not accurately reflect a user's true performance on quizzes. Since the calculation system is applied to every quiz, this would globally affect all grades, but not any other metrics.
b. Likelihood: 1 - The calculations used for grade computation are very simple and straightforward, leaving a very small chance for a calculation error to appear. Due to ad hoc testing, developers can almost be certain this bug doesn't appear.
c. Mitigation: If the grading scripts did not calculate quiz grades correctly, incorrect user grades could be reset in the database after the issue is fixed in the HTML code.

As this error has a low risk, and ad hoc testing has almost ensured a correct grading script, only a very small amount of unit testing (for testing the low level math) and system testing (testing a sample quiz to receive an expected grade) needs to be used.

B4. Incorrect database procedures
If a general database procedure fails to operate correctly, it could mean data is not transferred correctly, calculations are not correctly computed, etc.

a. Severity: 3 - Database procedures are essential to the base functionality for most of the Onboarding system. If one small procedure isn't working properly, it could break a whole unit of functionality of the website. For example, if the get_quiz_questions procedure wasn't functioning, users would not be able to take any quiz at all; they might not even be able to edit a quiz. Thus, a bug with a database procedure would result in very secure consequences.
b. Likelihood: 1 - The database procedures in this system were designed to be atomic and only accomplish single tasks. This makes them extremely readable and comprehensible. Because of this lack of complexity, here is a very low likelihood of any bugs appearing in the procedures.
c. Mitigation: If a faulty procedure was found, a system manager would temporarily shut the website off, and a manager who is experienced with MySQL would go into the database workbench and fix the offending procedure. This fix would theoretically be very easy because of the simple, atomic design of all procedures.

This part of the system has an overall medium risk. Since database procedures are such an important part of the base functionality, they should be tested using unit testing. Each individual procedure should be tested on its own given sample parameters. This ensures the foundation of the system is sound. Integration of the database procedures with the overall system has already been covered in the rest of the testing strategies mentioned previously (for example, deletion of a quiz, creation of users, etc).

8. Roles

System Owner: Josh Jackson, Michelle Darby
Developers: Victor Caceres, Matt Keville, Christopher Pham, Franklin Adair
Management: Professor Jack Myers
Testers: Developers, Scrum Master, Product Owner, Future system owner
Supporting Roles:

- **Scrum Master**: David McCullin
- **Product Owner**: Steven Douglass