5 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopted test strategy and instruments. In this overarching introduction, given an overview of the testing strategy and your team's overall testing philosophy. Emphasize any unique challenges to testing for your system/design.

In the sections below, describe specific methods for testing. You may include additional types of testing, if applicable to your design. If a particular type of testing is not applicable to your project, you must justify why you are not including it.

When writing your testing planning consider a few guidelines:

- Is our testing plan unique to our project? (It should be)
- Are you testing related to all requirements? For requirements you're not testing (e.g., cost related requirements) can you justify their exclusion?
- Is your testing plan comprehensive?
- When should you be testing? (In most cases, it's early and often, not at the end of the project)

5.1 Unit Testing

What units are being tested? How? Tools?

Each individual homework question is tested based on the inputs and the expected outputs. This testing is all done manually for each problem as each problem is different. The extensiveness of the test depends mostly on if the problem is parameterizable or static.

5.2 Interface Testing

What are the interfaces in your design? Discuss how the composition of two or more units (interfaces) are being tested. Tools?

The PrairieLearn framework handles most of the interface communication between modules. So we have had little to no testing in this area.

5.3 Integration Testing

What are the critical integration paths in your design? Justification for criticality may come from your requirements. How will they be tested? Tools?

The critical parts of our project are making sure the automated grading and question randomization work reliably and correctly. This fits our user's needs and lets us make sure the questions are being created correctly. This is being tested by creating a few variants and seeing if any errors come up, and if they do, they are fixed. The only tools we would use during this would be PrairieLearn itself.

5.4 System Testing

Describe system level testing strategy. What set of unit tests, interface tests, and integration tests suffice for system level testing? This should be closely tied to the requirements. Tools?

Our system testing is pretty simple since the only thing that we have to worry about is getting the server running with prairie learn on it. Prairie learn has everything else handled so once we make sure that it it up on the server with ISU authentication then we should be good.

5.5 Regression Testing

How are you ensuring that any new additions do not break the old functionality? What implemented critical features do you need to ensure do not break? Is it driven by requirements? Tools?

Our regression tests go with our unit tests and we will just have to double check them to make sure that they still work after we add anything. They are pretty straightforward and just need to make sure that each question works

5.6 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?
Since PrairieLearn handles most of the testing from the framework by itself, it will give us a good estimate of the problem so the problem wont affect the homeworks on the production side. If there is nothing wrong with the code then PrairieLearn will run fine and accept it no matter the outcome.

5.8 Results

What are the results of your testing? How do they ensure compliance with the requirements? Include figures and tables to explain your testing process better. A summary narrative concluding that your design is as intended is useful.

The results from our testing is that if something is wrong in our code for the homeworks information, question information, or questions logic then PrairieLearn will give us an error screen with what is wrong and why. This is very helpful as we are making many variants to complicated questions and something unexpected could happen with just one of the many in our pool of questions. I do not think that our testing design is the greatest that it could be, but because of our restriction of using PrairieLearn and because it handles so much of errors internally, there is not much room for us to make a better design.