Test Plan

1. Test philosophy

a) Could you test everything you can think of? What criteria are you using to select what gets tested?

Yes we could test everything we can think of. The criteria we are using to select what gets tested is based on the user’s perspectives of their activities

b) Are there different densities of testing across your code? Will less well understood code go through more testing? Is there a code that does not require testing?

There are different densities of testing across our code. For example, we would test heavily on the primary functions like joining/creating an event and the machine learning component. We would focus less on more simple portions of the applications such as logging in/out, create account, modify settings, etc. For less understood codes, they will be tested more than simple codes but not as much compared to codes for like machine learning.

c) Why would your selected testing program be sufficient? What criteria do you use for making that decision?

We believe that the selected testing program would be sufficient due to the coverage of priority. The testing will cover all of our components in our application but with different priorities. For example, main functions and components will be tested more than smaller functions like UI. These are part of the criteria that we use to make our decision. The criteria would be to test everything but through a priority-based system.

2. Team workflow

a) What is your team’s test process?

Finish a function, then test it. Do this repeatedly until all of the functions are done. We will then connect those functions to each other and test the whole application.

b) How do you determine code coverage?

We first decide who wants to do what. Then, we will cover the untaken tasks with other teammates. Since our team does not really care so much on what portion of tasks they are assigned, dividing the task is fairly simple.

c) Do bugs come back from users or from your own team affect your test program workflow?

Bugs would come back from users and from our own team. While users would report them to us, our team will constantly monitor any bugs throughout developing our web application. These bugs will not affect our test program workflow since most bugs will be anticipated. For those that might affect our testing workflow, we believe that it can be mitigated due to our team’s knowledge of their work.

3. Test design

a) Code paths

Unsure on how to answer this, but from what is suggested from an online source is to use Basic Path Testing. It is based on a White box testing method that defines test cases on the flows or logical paths that can be taken through the program. It involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid of branch testing and path testing methods. To do this, we must first draw a control graph, calculate the cyclomatic complexity, find a basis set of paths, then generate test cases to exercise each path. The advantages for using basic path testing path testing are that it helps to reduce redundant tests, focuses attention on program logic, helps facilitates analytical versus arbitrary case design, and for each test cases, execute every statement in a program at least once.

b) Code coverage estimation in terms of % lines of code tested (related to (a) above)

Unsure on how to answer this but from what we have found online, we will be using a formula: Tcov = (Ltc / Lcode) * 100%, where Tcov = test coverage, Ltc = number of lines of code covered by tests, and Lcode = the total number of code lines. From our online research, they suggest using a toolkit called Clover that allows you to analyze which lines were entered during testing so that we can significantly increase the coverage by adding new tests for specific cases. In addition, it can get rid of duplicate tests.

c) Environment

Unsure on how to answer this, but for what we have found online suggest that we should determine if test environment needs archiving in order to take back ups, verify the network configuration, identify the required server operating system, database, and other components, and Identify the number of license required by the test team.

d) Test code and data

Unsure on how to answer this, but online sources suggest using given softwares to test our code and data. For example, python has several testing softwares to use. These include, unittest, doctest, py.test, hypothesis, etc. We need to learn more about this to answer this question completely and correctly.

e) Version control methodology (of your product code, test code and test drivers. Do you track bugs?)

For what we can understand, we will be using GitHub to simultaneously work on the same version without hindering each other’s work. For every build we release, we will try to find bugs from our own testing or from other user’s. We will then try to fix those bugs and release an updated version of the web application.

4. For every test case: *(The following can be part of your test code, if organized clearly and well)*

a) Requirements or behaviors, i.e. what your tests are trying to verify (requirements) or validate (user needs) for every test – Is there a more systematic way to establish this?

For every test case, our requirements will be whether it works or not. It is still too early for us to know what all our requirements and behaviors are. However, for what we know right now is that our test cases will primarily focus on whether or not our goals will be reached. For example, test functionalities for all of the required portions of the web application. If they the testing goes well for those, move on to the other required functionalities.

b) Evaluation criteria (e.g. pass/fail) and disposition

Similarly to the questions above, we are unsure on how to answer this with what we know right now. However, in the simplest way to answer this, we will evaluate our program through if it works or not. These would primarily be focused on the functionalities that are must have in our applications. If it is what we desired and it works, then it passes. If it is what we desired, but does not work, it fails and needs to be worked on.