A. 20% Effective use of project design activities as documented in your:

a. Overall project description

b. Backlog of user stories

c. UI sketches or wireframes

d. UML diagrams

(which may include class, sequence, activity or other diagrams)

B. 20% Effective project management and team collaboration as observed in your:

a. Agile backlog, sprints, and burn-down reports

b. Git pull requests and commit logs

C. 40% Effective implementation of your project as seen in:

a. The quality of your code

(did you follow good design and coding practices?)

b. Completeness of your unit tests

c. The completeness of the final product

(how many and which features were implemented?)

d. Satisfaction of your client and/or feedback from testers

D. 20% Your final project presentation and documentation; which should include:

a. Revised documentation

(revise the original design documents to reflect any changes or additions.)

b. Evaluation of the development process. Describe what went well and how you could do each

of these things better in the future:

(Discuss these things in your presentation- putting them in written form is optional)

For web developers: Communication with your client, especially elicitation of requirements.

For Game developers: Clarity of your own requirements for the game.

ii.Team collaboration

iii.Designing the software

iv.Project management

c. Evaluation of the finished product:

i. Assessment of the completeness and correctness of your system

1. Is the software done (and how do you know what “done” means)?

2. For web developers: Does it do what the client wants it to do? For game developers: Did it turn out the way you wanted it to?

ii. Assessment of the quality of your design:

1. Were there any features that were especially hard to implement because of

the design?

2. Is this design scalable (will it be easy to add features)?

3. Is the design easy to understand and communicate to others?

4. Does this design result in sufficient performance (will the web site or game be

responsive in the more challenging use cases, such as when it is run an

older/slower machine, when there are many simultaneous users, when it is

performing computationally intense operations?

5. In retrospect, would a different design have been better?

iii. Assessment of the implementation of your system. Note that some of these questions may seem redundant to the questions in section II, design, but these questions are about the consequences of the way you did your coding, not the design of your software. (For each question give an explanation, not just a yes or no answer)

1. Will the code be easy to maintain?

a. Is it easy to fix bugs?

b. Will it be easy to add new features to the system (how do you

know)?

c. To what degree did you follow accepted coding practices (separation

of concerns, DRY code, no “dead” code, minimal dependencies)? Give

examples.

2. Is the code well documented internally?

a. Descriptive names for classes, methods, and variables

b. Comments where needed to explain what your code is supposed to

do

c. Are there unit tests that make the purpose of methods explicit?

3. Will your code run fast (for example, will your database support a heavy load of users, or will large numbers of moving objects be rendered smoothly)?

4. Is your code efficient (did you accomplish your goal without writing a lot of extra code)?

5. How bug-free is your code (are there known bugs, does it crash unexpectedly, or does it do the wrong thing at random times)?

6. How do you handle run-time exceptions?