**Testing #2 - Whiteboard Interviewer**

In the V1 released, we focused on bug fixes and other QA related issues.

You can check out our issue tracker on Git for a complete picture. We have QUnit for JS testing on our repo, although we haven’t written a lot of tests for it. Also, our PHP has unit testing which you can see in our repo as well. We have a few frontend related features that is hard to automate as tests, in which case we wrote “unit tests” that a human can verify. These are on our issue tracker. Since the previous testing phase, we’ve outline a convention on how to report bugs in the issue tracker. The convention includes indicating the severity, description, how to reproduce, and who the bug is assigned to. We also refactored and improve the stylistic aspects of our code.

**Coding Standards and Guidelines for Programming Languages**

Coding Standards / Guidelines seems to only have benefits, rather than disadvantages. The only possible disadvantage is the overhead

of establishing and enforcing these standards. However, the time saved from implementing these standards outweigh the time loss in the overhead.

One aspect of Coding Standards is consistency. Engineers do not have to reacquaint themselves reading the code for separate modules developed

by separate people. Like the "broken window theory", coding standards will also improve code smells and just the general attitudes developers will have for the project. Coding standards will make life much easier for maintenance, where a lot of time can be loss in software projects on maintenance. If we accept that coding is essentially writing, then we want our writing to be professional and clean as possible. Furthermore, it is a matter of

Efficiency. For our project, we did not have a formal document. If we had more time, this is a totally a feasible and worthwhile option,

**Static Analysis Tools**

These definitely save developer time, and will reduce bugs. Using this should be an "aid" rather than a be-all-end all solution. They should not use this tool and presume that everything is correct. It is merely another layer in program correctness, and is best used with other techniques (tests, etc.). Integration of these tools into a project may be complex and a huge overhaul. The degree to which such tools can be incorporated, and utilized efficiently relies on the context of the project. For our project, personally, we think integrating this tool might be too complicated. Our control flow is very simple, and does not need have the degrees of freedom that would take full advantage of static analysis tools. That is, the overhead to integrate this tool is not outweighed by the advantages.

**Code Review Checklist**

Code review checklists would provide consistency in code reviews. It might not be so much useful for senior developers, who already have the checklist in their heads. It would, however, be helpful to new members on a team / project. Of course, this might be a double edged sword. If people abide solely by the checklist and don't utilize their common sense / knowledge, then they might miss out on important stuff. Moreover, there may be special scenarios that are not covered in a CR checklist intended for general purpose use. If the CR checklist is held important, then the quality of the CR's depend strongly on the quality of the CR checklist. In a way, this is analogous to the design of a software architecture. The quality of each component is measured by prior planning. Another aspect of CR's is that the reviewer needs to have a mild degree of knowledge about what she's reviewing. For our project, the frontend developer did not know the details about the backend developer's component. We believe it would have cost a lot of time to have cross component CR's. However, this might have been useful in cases where people were collaborating on a component. There were cases where some bad code would slip through, but the majority of the time our developers produced quality code. For a company with large resources, where bugs in their code will have widespread impact, they definitely need to emphasize CR. For our project, we don't know if we'd be able to hit all the project milestones if we were bogged down with all these meta-programming related issues.

We respect these methods for quality assurance in software, although we did not incorporate all of these components into our project.

The main reason is the time constraints. The overhead of incorporating these methodologies is worthwhile for longstanding projects, and companies with resources (one resource being Senior Software developers’ expertise and knowledge). That is not to say we did not extract some useful bits from each of these methods. We implicitly applied principles reflected in these methodologies, albeit without the formalities / documents. For example, we adhere

to coding standards taught to us by the CS component, and we would have people read each other’s commits to Git and provide feedback (again, informally).