**CST316 Software Enterprise: Construction and Transition Spring 2016 f2f**

**Lab: Code Inspection Exercise February 17, 2016**

**Objective:** Understand how to conduct a collaborative code review.

**Activity: Perform Code Review**

*Step 1: Setup*.

* Work in pairs.
* Grab starting the code zipfile off of Blackboard, and the MS-Word Code Review Defect Log form.

*Step 2: Individual review of the code*. **YOU SHOULD BE DONE BY 2:15pm!**

* Create a new repo named cst316labcr under 1 person on Github. Push the code I gave you up to this new repo. Add your partner, plus github user cst316, as collaborators.
* Review the source code files individually for conformance to the coding standards provided on the next page, for general quality practices, and for logic errors. Log any defects using the MS-Word form. Make sure your individual name is at the top and the form is named crlab\_step2\_<asurite>.docx, you should be done with this by about 2:10 and turn it on Blackboard (or by hardcopy) by the end of class.
  + On the form at the top is a place to enter the GitHub repo. We need this so we know where to go to find your GitHub work to grade the lab. I might put “cst316/cst316labcr” here for example. DON’T FORGET!

*Step 3: Review defects together*. **DONE BY 2:45pm – we will pull your GitHub repo!**

* Review your defect lists together. Merge the defect lists into one master list with agreed upon category and severity descriptors. Enter your defects found on GH in the Issues tool. Label each with a severity (BLOCKER = RED, MAJOR = BLUE, LOW = YELLOW). When entering the review item on GH as an Issue, make an initial comment that includes the description, which defects from your individual lists it is derived from, and the agreed upon Category. Example:
  + My problem #3 and Suhas’ problem #7 both refer to the fact that file foo.java is missing a header comment at the top of the file. We create a GH issue, label it LOW (yellow), and in the initial comment summarize the problem description, put in “kgary-3” and “sxavier2-7”, and indicate category CG.
* Assign the issues found to each individual. Each person should take on 1 defect from each category listed on the Code Review Defect Checklist. You should select defects of higher Severity.

AT THIS POINT WHEN YOU LEAVE CLASS TODAY YOU SHOULD HAVE SUBMITTED A WORD DOCUMENT WITH YOUR INDIVIDUAL CODE REVIEW DEFECT LIST, AND IT SHOULD INDICATE YOUR GITHUB REPO. FURTHER, YOU CAN USE THE TEXTBOX ON YOUR BLACKBOARD SUBMISSION TO PUT YOUR GITHUB REPO IN AS WELL. WE WILL THEN PULL YOUR REPO AND YOUR ISSUES!

*Step 4: Fix defects outside of class and use the Git Pull Workflow*

* Create a branch (not a fork!) in the repository for each defect, name it “asurite-#” where asurite is the ASURITE if of the person who is fixing the defect and # is the Github issue number.
* Correct the defects assigned to you and when each one is done issue a pull request to master.
* Your partner is responsible for reviewing the changes, making comments and finishing the pull request
* As a reviewer, you MUST REJECT AT LEAST ONE PULL REQUEST and make your partner go back and try again!

**DONE WITH THIS STEP BY MONDAY AT 11:59PM! We will be pulling your repos then and inspecting logs!**

**Grading**: The code given compiles and runs but has several faults. To motivate you to find and log defects, your grade will be in part determined by how many of the defects you find that I have seeded throughout the code. Another part is the extent to which you improve the internal code quality of the code. Below is a Coding Standard the code should follow, but keep in mind violation of the coding standards are only one type of defect (see the Category description at the bottom of the Code Review Defect List).

# Expectation for your projects going forward:

I expect that from this point forward you will incorporate code reviews into your quality policy:

1. Decide on when and where and why you will do reviews. Which code? How many people? What level of formality?
2. Define how you will categorize issues found – level of severity, and type of defect. How will you indicate that?
3. Define a coding standard. Quality codebases look like they are written by one person.
4. I expect you will GitHub issues to track code review defects, and you will have a Taiga task for code reviews under the User Story of the code you are writing.
5. Make sure in your process and tool it is clear whose code is getting reviewed, and by whom it is getting reviewed. You should serve as both an author and a reviewer in your projects for one formal and one informal review (4 reviews total at a minimum).

# Coding Standards

1. All source code files must have a file banner comment present and filled in. This banner is available in the templates.java file on Moodle.
2. All public classes must have a class banner comment present and filled in. This banner is available in the templates.java file.
3. All public methods, except getter/setter methods, must have a method banner comment present and filled in. This banner is available in the templates.java file.
4. Naming conventions are as follows:
   1. Constants and Enums should be in all CAPS (example: PI)
   2. Class names should be upper CamelCase, with the first letter uppercase (example: MyClass).
   3. Variable, Parameter, and Method names should be in lower camelCase, with the first letter in lowercase (example: fooBar).
   4. Non-public methods, variables, and constants should be prefixed by a leading underscore \_.
5. All attributes must be private (class member variables, not constants).
6. All literal values, except loop indices starting at 0 or 1 must be declared as constants.
7. Classes should list methods and attributes in the following order, top-to-bottom:
   1. Constants
   2. Constructors
   3. Public setter/getter methods in alphabetic order
   4. Public non-getter/setter methods
   5. Private methods
   6. Private class member variables
8. All code should be consistent stylistically. This includes:
   1. All {} should appear with the { at the end of a line and } on its own line
   2. Indentation should be consistent.
   3. All complex statements (if, else, switch, loops) must use explicit {} even if the body is a single line.