Rob Wagner, Alex Petitjean, and Ryne Bell

Nutrition App Metrics

1. Lines of code (class / method)

- Our team is going to keep track of the number of lines of executable code plus the data definitions. We are going to aim for our methods to have less than 50 lines of code and our classes under 750, as is standard.
- We are going to use the Metrics plugin for eclipse to monitor this.
- We plan to use it to analyze how much code we are adding to the system when a new feature is implemented, and how much we are removing / consolidating during a refactor.
- We want to be able to see how quickly our system is growing and maybe where it needs refactoring. We want our system to be clean and streamlined, so we will want to eliminate clutter.

2. Afferent / efferent coupling

- Afferent and efferent coupling allow us to measure how intertwined our classes are. Afferent coupling measures how many packages depend upon classes within the current package, and efferent coupling measures how many packages the current class depends on.
- Luckily, the Metrics plugin can show us the coupling for each package!
- We will use this to see how essential a package is and how dependent it is on other packages in the system.
- Hopefully this will help us reduce/refactor the system if we can see that a package has too much dependent on it, or perhaps not enough.

3. Nested block depth

- We will measure how many nested loops each block of code has, if it is more than 5 we know that we need to break up the method.
- The Metrics plugin nicely shows us how nested each block is.
- We want to use this to analyze the complexity of our code. We don't want it to end up being too complicated to pass on.
- When we are traversing an image by pixels and creating transition matrices we end up looping through a lot of data. We need to monitor this so that we don't end up making our code too complicated.

4. Data on fixes

- We are going to keep track of current problems in the system using a simple log on our github.
- If after a coding session a problem exists, it will be added to our fixlog with the date and time of its discovery. This way we can measure how fast we fix these problems, how many delinquent problems we have at any time, and how well do we fix bugs (does it pop up again later?).
- We will use this data to monitor how quickly we were able to handle problems and give some measure of the quality of our fixes.
- We want to be able to improve how quickly we handle bugs, and we also want to be able to improve of fix quality. We need to try to develop our fixes so that it won't pop up later in the development process.