Gavin Scott

February 12th, 2016

Will Code for Food

**Metric:** Abstractness vs. Instability

Note: Only non-UI, non-test packages were considered

**Data:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package** | **Abstractness** | **Instability** | **Sum (A + I)** |
| model.common.messaging | 0.33 | 0.57 | 0.9 |
| model.common.form | 0.2 | 0.32 | 0.52 |
| controller.retrieval | 0.17 | 0.25 | 0.42 |
| model.common.common | 0.11 | 0.26 | 0.37 |
| model.getInvolved | 0.0 | 0.4 | 0.4 |
| model.ridesharing | 0.0 | 0.68 | 0.68 |
| model.common.common.users | 0.0 | 0.75 | 0.75 |
| tasks | 0.0 | 0.77 | 0.77 |
| controller.api\_interfaces | 0.0 | 0.6 | 0.6 |
| model.resources | 0.0 | 0.25 | 0.25 |
| controller | 0.0 | 0.33 | 0.33 |
| model.common.common.sorting | 0.0 | 1.0 | 1 |
| controller.authentication | 0.0 | 0.75 | 0.75 |
| values | 0.0 | 0.0 | 0 |
|  |  |  |  |
| **Average** | **0.089** | **0.4** | **0.55** |

**Analysis:**

Instability is measured as the percentage of the total coupling (both efferent and afferent) that is efferent:

Essentially, if a module is completely stable (instability = 0), it depends on no classes outside it’s module, and all coupling is from dependencies going the other direction. Conversely, a module is completely unstable when no other modules rely on it, and it relies on others. Abstractness is calculated by measuring the percentage of classes in a module that are abstract.

Looking at the balance between instability and abstractness is useful to measure the quality of the design of our code. Ideally, code that is heavily relied upon by other modules should be abstract, and code that is not abstract should not be relied upon too heavily. In other words, code should rely on abstractions rather than implementations. This prevents minor changes in implementation from creating errors that propagate throughout the program and are potentially difficult to track down and fix. So, the sum of the instability and abstractness for each module should be somewhere near 1 (meaning that the stability and abstractness of a module are roughly the same).

Examining our code, some modules are very well balanced. The messaging and sorting modules in particular are very close to having equal stability (1 - instability) and abstractness. Many of the other modules are acceptable (ridesharing, users, tasks, and authentication), but these could be improved. The others, particularly the heavily-used “common” package, have a poor balance and is potentially a source for a large number of future errors if changes to it are not made very carefully.

The fault with these metrics is that they are module-specific. This is a benefit because it allows us to calculate coupling based only on classes outside a class’ package and ignore coupling internal to a package (which does not decrease modularity). However, it prevents us from identifying which classes are being relied on; it is possible that we could have an even balance of stability and abstractness at a module level, but all incoming dependencies are dependent on the implementations of the abstract classes, rather than the other way around. I looked through our code and did not find this to be the case, but in general we could improve the quality of our code design by consciously avoiding relying on implementations of an abstract class, and rely on the abstract class instead. This could be as simple as writing functions to accept the abstract superclass of a type of object rather than a specific implementation. If this is not possible, and a class needs to rely on a specific implementation, it may make sense to consider refactoring so that the implementation and the dependent class are in the same package. If such a strong dependent link between them exists, letting them remain in separate packages reduces the modularity of the project, and hurts our design.

These metrics should not be too difficult to improve with some refactoring, although reaching an ideal state will likely not be possible, and the better the metrics become the harder they will be to improve.