Chidamber and Kemerer Metrics

The metric set Chidamber & Kemerer is an object-oriented metrics suite that presents 6 metrics. The metrics exhibited by this metric suit are:

# Coupling Between Objects (CBO)

This metric indicates the number of classes to which a class is coupled. The instances of coupling considered for this metric are method calls, field accesses, inheritance, arguments, return types and exceptions. This metric can help identify eventual cases of the Code Smell Inappropriate Intimacy as classes with high coupling are more likely to be dependent on others and of establishing a cycle of dependency.

This metric locates eventual trouble spots, such as the class FreeColObject with whom there might be many strong dependencies.

# Depth of Inheritance Tree (DIT)

This metric presents the maximum inheritance path from a given class to its root. Therefore, this metric can aid the identification of the Code Smell of Refused Request, as classes with large inheritance paths might be, for instance, inheriting unnecessary methods. A large inheritance path may also indicate a problem of Speculative Generality, as a too-general parent class would lead to the need of more children classes with implementations that actually tackle the problem at hand.

Classes with high inheritance paths that could, therefore, be identified through this metric as potential trouble spots are, for example, the classes ReportCargoPanel, ReportMilitaryPanel and ReportNavalPanel.

# Lack of Cohesion of Methods (LCOM)

This metric counts the sets of methods in a class that aren't related through the sharing of some of the class’ fields. This metric has since suffered many alterations after being met with much critique, so it wasn’t employed in the identification of potential trouble spots.

# Number of Children (NOC)

This metric presents the number of immediate descendants of a given class. A high number of children indicates a heavy reuse of the class in question which, on the one hand, might be beneficial as it allows for code reuse, but it also may indicate problems in the abstraction of the parent class. The latter might result in the Code Smell of Speculative Generality, as the parent class could have been made to tackle future problems, instead of the immediate ones, which then result in the need for more children classes.

Classes with very high NOCs may be identified as potential trouble spots, such as the class FreeColAction, which might be overly general in order to then warrant such a necessary number of children.

# Response for Class (RFC)

This metric presents the number of distinct methods that can be executed when an object of that class receives a message, that is, when a method is invoked for that object. This metric can assist in the location of the Code Smells of Feature Envy and Inappropriate Intimacy. In the case of Feature Envy various instances of communication between methods of different classes might indicate that this code should be together. A case of Inappropriate Intimacy might arise from a dependency established through the communication between the classes’ methods.

Classes such as InGameController from the Client package, which holds a very high RFC (700) could be considered potential trouble spots for the above mentioned Code Smells.

# Weighted Methods Per Class (WMC)

This metric counts the number of methods in a given class. The high number of methods suggest that some classes might present the Code Smell of Divergent Class, as they are taking on many responsibilities that then warrant the presence of so many methods. This metric can also indicate the Code Smell of Large Classes, as classes with many methods tend to be longer.

Through analysing this metric we can identify potential trouble spots, such as the class InGameController which holds a very high number of methods (786), therefore signifying an eventual divergence in the present methods as the class holds too much responsibility.

# Relation to Code Smells

Duplicated Code

By analysing the metric WMC we were able to locate the Code Smell Duplicated Code in class Colony, as the number it displayed for this metric was quite high (396). The elevated number of methods indicated the eventual presence of unnecessary code, such as the duplicated methods we identified.

Message Chains

Through examining the metric RFC we were able to detect the Code Smell Message Chains in class ServerColony, as the number it exhibited for this metric was quite high (209). The elevated number of methods being executed as a method is invoked for that object indicated the eventual occurrence of chaining.

Long Method

By inspecting the metric WMC we were able to identify the Code Smell Long Method in class ServerPlayer, as the number shown for this metric was quite high (601). As the class with the 3rd highest number for this metric, there was a high likelihood of locating Code Smells, making this class trouble spot.

Beatriz de Caiola Machado, 62551