Lines Of Code

* L(J) – linie kodu w Javie
* L(XML) – linie kodu w XML
* LOC(Lines of Code) – ogólna liczba linijek kodu ( z komentarzami)
* NLOC – liczba linijek kodu bez komentarzy

Całość naszego projektu zawiera 2995 linijek kodu, co świadczy o dość dużym rozmiarze projektu. W tym 2637 linijek kodu jest napisana w Javie zaś aż 354 w XML, który opisuje okienka interfejsu. Świadczy to o położeniu dużego nacisku na interfejs i poprawną komunikację z użytkownikiem.

Najwięcej kodu posiada pakiet Controllers (1062 linijek kodu), który obsługuje wyświetlane okienka. Ma ona za zadanie pobrać dane wpisane przez użytkownika, sprawdzić ich poprawność, przesłać spreparowane dane dalej oraz wyświetlić informacje użytkownikowi. Kolejno następne pozycje zajmują moduły Modelu (617 linijek), DB Contextu (534 linijek), Repositories(331 linijek) i Utils(48 linijek).

MOOD Metrics

(Metrics for Object-Oriented Design). Metryki te mają służyć całościowej ocenie systemu, a wyrażone są w procentach oznaczających stopień wykorzystania mechanizmów charakterystycznych dla programowania obiektowego. Metryki MOOD mierzą następujące elementy:

* + polimorfizm Polymorphism Factor (PF)
  + hermetyzacja Attribute Hiding Factor (AHF) Method Hiding Factor (MHF)
  + dziedziczenie Attribute Inheritance Factor (AIF) Method Inheritance Factor (MIF)
  + przekazywanie komunikatów Coupling Factor (CF).

MHF w naszym projekcie wynosi 18%, co znajduje się w pożądanym przedziale 10-25%. Część metod wewnątrz klasy nie jest dostępna na zewnątrz.

AHF wynosi 86,5% - co oznacza, że 13,5% atrybutów w naszym projekcie posiada publiczne modyfikatory dostępu.

AIF – wynosi 15% co oznacza, że w projekcie niezbyt często korzystaliśmy z dziedziczenia atrybutów. Specyfikacja naszego projektu nie ułatwiała osiągnięcie wysokiego współczynnika AIF. Przy realizacji większej ilości przypadków użycia na stworzonym systemie współczynnik AIF byłby zapewne większy.

MIF – wynosi 10% - sytuacja jest analogiczna jak ze współczynnikiem AIF

PF – wynosi 79% - co oznacza, że aktualny system ma skomplikowane hierarchie dziedziczenia. Związane jest to z zastosowaniem wzorca architektonicznego MVC.

CF – wynosi 24,09% - wartość przekracza optymalny zakres. Oznacza to, że w nasz systemie występuje wiele powiązań między klasami innych niż dziedziczenie. Związane jest to m.in. z przekazywaniem referencji do kontrolera przez metodę otworzOkno w klasie ViewControler, dzięki której uzyskaliśmy ustandaryzowaną formę często wykonywanej akcji. W teorii oznacza to, że nasz system jest mało elastyczny i trudny w modyfikacji i pielęgnacji.

The Chidamber and Kemerer Metrics

* CBO - Coupling between Objects
* DIT - Depth of Inheritance Tree
* LCOM - Lack of Cohesion of Methods
* NOC - Number of Children
* RFC - Response For Class
* WMC - Weighted Methods for Class

**Overview**

* CBO - Straightforward calculation. Low values are good.
* DIT - In general, but not always, a high DIT is viewed as a good thing - so you can use this as an indicator and then use your judgement after a visual examination of the class. There is some debate about calculation and, in the case of Java, what do we do about interfaces?
* LCOM - Wide variety of measurement approaches. Not sure what it indicates. In general high levels are viewed to be bad
* NOC - A high level may be either a good or a bad thing, potential for confusion in measurement, particularly with regard to interfaces
* RFC - A high level is a useful indicator of potential problems, uncontroversial measurement
* WMC - Doesn't really do what it claims to do - replace with either number of methods or total cyclomatic complexity

Complexity metric

* OCavg - Calculates the average cyclomatic complexity of the non-abstract methods in each class. Inherited methods are not counted for purposes of this metric.
* WMC - Calculates the total cyclomatic complexity of the methods in each class.
* ev(G) - Calculates the essential complexity of each non-abstract method. Essential complexity is a graph-theoretic measure of just how ill-structured a method's control flow is.
* iv(G) - Calculates the design complexity of a method. The design complexity is related to how interlinked a methods control flow is with calls to other methods.
* v(G) - Calculates the cyclomatic complexity of each non-abstract method. Cyclomatic complexity is a measure of the number of distinct execution paths through each method.