Empirically Validating an Analytical Method for Assessing the Impact of Design Patterns on Software Quality

Ampatzoglou, Apostolos; Avgeriou, Paris; Arvanitou, Elvira Maria; Stamelos, Ioannis

Empirically Validating an Analytical Method for Assessing the Impact of Design Patterns on Software Quality: Three Case Studies

APOSTOLOS AMPATZOGLOU, University of Groningen, the Netherlands

PARIS AVGERIOU, University of Groningen, the Netherlands

ELVIRA MARIA ARVANITOU, University of Groningen, the Netherlands

IOANNIS STAMELOS, Aristotle University of Thessaloniki, Greece

rijksuniversiteit groningen

Α.Π.Θ. A.U.Th.

Contents

Abstract

This technical report has been created as support material for the paper entitled "Empirically Validating an Analytical Method for Assessing the Impact of Design Patterns on Software Quality: A Case Study" that has been submitted in ACM Transactions on Software Engineering. The corresponding paper aims at validating an analytical approach that can be used for comparing object-oriented design structures. In this technical report we present in detail the three case studies that are reported in the paper. The references of the technical report correspond to the papers reference list.

1. Design Quality Metrics

In [Bansiya and Davis 2002], the authors propose a hierarchical quality model that aims at quantifying six design quality attributes from measurements on object-oriented design components. The design quality attributes that are involved in the model are reusability, flexibility, understandability, functionality, extendibility and effectiveness. The exact definitions of the six design quality attributes can be found in [Bansiya and Davis 2002]. The object-oriented design properties that are used in the model are design size, hierarchies, abstractions, encapsulation, coupling, cohesion, composition, inheritance, polymorphism, messaging and complexity [Bansiya and Davis 2002]. In addition to that, the model employs several object-oriented design metrics in order to measure the aforementioned properties. Finally, the components that can be identified in a design in order to measure their properties are classes, objects and relationships between them.

Furthermore, in [Bansiya and Davis 2002] the authors provide several links for mapping attributes of a lower level to a higher one. The final outcome of mapping attributes is six mathematical statements that map the object-oriented design metrics to the aforementioned design quality attributes. As mentioned above, the QMOOD model involves eleven (11) object-oriented design properties each one quantified through one object-oriented design metric [Bansiya and Davis 2002].

- "Design Size" property - DSC (Design Size in Classes) metric. This metric is a count of the total number of classes in the design. Range of values [0, +∞)
- "Hierarchies" property - NOH (Number of Hierarchies) metric. This metrics is a count of the number of class hierarchies in the design. Next, the "Abstraction" property is measured through the ANA (Average Number of Ancestors) metric, which signifies the average number of classes from which a class inherits information. Range of values [0, +∞)
- "Encapsulation" property - DAM (Data Access Metric) metric. This metric is the ratio of the number of private attributes to the total number of attributes. Range of values [0, 1]
- "Coupling" property - DCC (Direct Class Coupling) metric. This metric is a count of the

different number of classes that a class is directly related to. Direct relations are considered to be attribute declarations and message passing in methods. Range of values [0, +∞)

- "Cohesion" property - CAM (Cohesion Among Methods of Class) metric. The metric computes the relation among methods of a class based upon the parameter list of the methods. Range of values [0, 1]
- "Composition" property - MOA (Measure of Aggregation) metric. This metric counts the number of data declarations whose types are user defined classes. Range of values [0, +∞)
- "Inheritance" property - MFA (Measure of Functional Abstraction) metric. This metric, is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class. Range of values [0, 1]
- "Polymorphism" property - NOP (Number of Polymorphic Methods) metric. This metric counts the methods that can exhibit polymorphic behavior. Range of values [0, +∞)
- "Messaging" property - CIS (Class Interface Size) metric. This metric is a count of the number of public methods in a class. Range of values [0, +∞)
- "Complexity" property - NOM (Number of Methods) metric. This metric is a count of all the methods defined in a class. Range of values [0, +∞)

The majority of the metrics are calculated at class level. In order to avoid correlations between the independent variables of our study, we have used the average function so as to aggregate the results at system level. Had we used summation, all variables would be correlated to the DSC metric.

2. Decorator

The aim of this section is to present the results of performing the enhanced analytical method on the Decorator pattern. Decorator is used when "you want to add behavior or state to individual objects at run-time" [Gamma et al. 1995]. In section 2.1 we present the structure on the Decorator pattern and two alternatives design solutions. In section 2.2 we present the results of applying the method.

2.1 Design Solutions

The class diagram of a typical Decorator instance is presented in Figure 1. The alternative design solution is presented in Figure 2. In the Decorator design pattern we have identified four axes of change, based on two class hierarchies and one pattern-related method.

Hierarchies:

- *Let n to be the number of Leafs in the design.*
- *Let p to be the number of ConcreteDecoratorA (those that provide additional methods than the ones provided by the given methods of the hierarchy)*
- *Let q to be the number of ConcreteDecoratorB (those that only exhibit different behavior on the given methods of the hierarchy, without providing additional methods)*

Methods:

- *Let m to be the number of operation methods, i.e. the number of abstract methods in the decorator class hierarchy.*

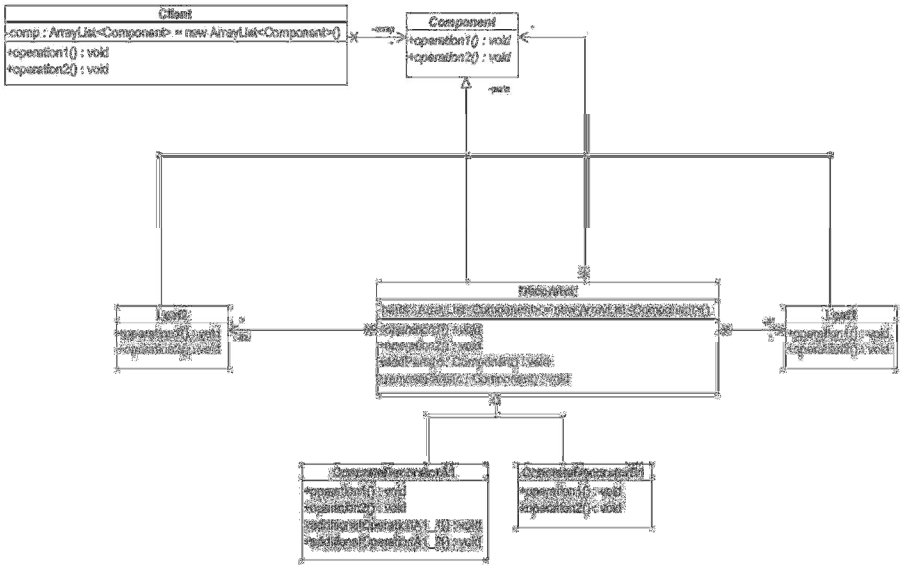

Figure 1. Decorator Design Pattern Class Diagram

The *Decorator* alternative design holds different array lists for each type of *Leaf*, in order to provide equal functionality on the aggregation to *Component* class in the design pattern. In order for the decorator to change type during run-time, the *Decorator* class holds a decoratorType attribute that can take (p+q) possible values. In this design, inside the m operations, we have placed (p) if statements, in order to handle all possible implementations of *Concrete Decorators*.

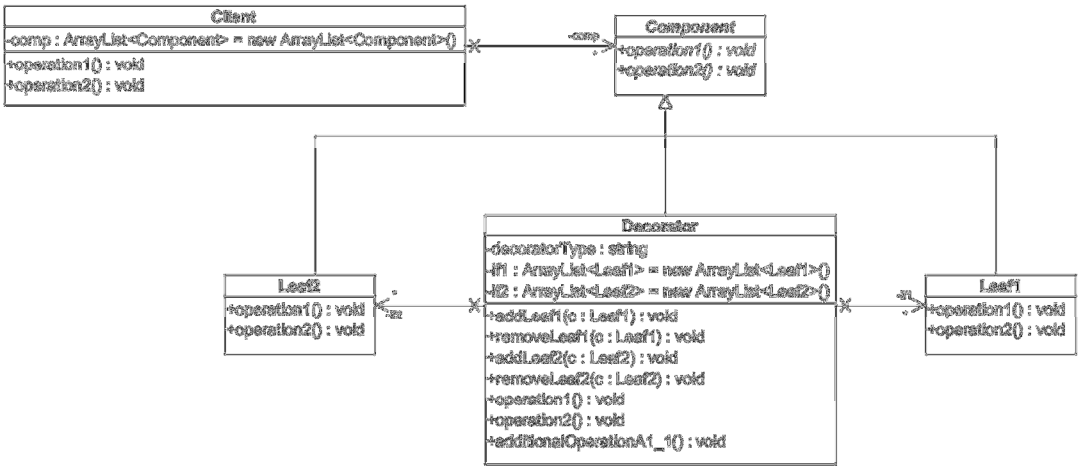

Figure 2. Decorator Design Alternative Class Diagram

2.2 Results

By taking into account the identified axes of change and the definition of the used metrics, we create the follow functions:

Pattern Solution

The number of classes in the system is the sum of the number of *Leaf*$_i$ classes (n), the number of *ConcreDecoratorA*$_i$ classes (p), the number of *ConcreDecoratorB*$_i$ classes (q), plus 3 (*Decorator*, *Component* and *Client*). Thus,

$$DSC = 3 + n + p + q$$

The NOH in classes *Component* and *Decorator* equals 1, because they inherit from other classes, at the first level. The other classes do not inherit from any others, so their NOH equals 0. Thus,

$$NOH = 2$$

The (p) *ConcreDecoratorA*$_i$ classes do not inherit two methods, i.e. addParts(c) and removeParts(c), from the *Decorator* class, so its MFA equals $(\frac{2}{2*m+2})$. The (q) *ConcreDecoratorB*$_i$ classes also do not inherit the same two methods from the *Decorator* class, so its MFA equals $(\frac{2}{2*m+2})$. Thus,

$$\text{MFA} = \frac{\frac{2}{(2*m)+2} + \frac{2}{m+2}}{3+n+p+q}$$

The *Client* class includes an object, of type *Component*, so its DCC equals 1. The *Component* class is abstract and does not reference any other object, so its DCC equals 0. The *Decorator* class includes an object type *Component*, so its DCC equals 1. The (n) Leaf$_i$ classes inherit from the *Component* class, so their DCC equals 1. The (p) *ConcreDecoratorA*$_i$ classes inherit from the *Decorator* class, so their DCC equals 1, whereas the (q) *ConcreDecoratorB*$_i$ classes inherit from the *Decorator* class, so their DCC equals 1. Thus,

$$DCC = \frac{2 + (1*n) + (1*p) + (1*q)}{3+n+p+q}$$

The *Decorator* class has one parameter type and (m+2) methods, thus its CAM equals $(\frac{2}{m+2})$. For the other classes CAM is not defined.

$$CAM = \frac{2}{m+2}$$

The *Decorator* class includes an object of type Component, so its MOA equals 1. The *Client* class includes an object, of type Component, so its MOA equals 1. All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

$$MOA = \frac{2}{3 + n + p + q}$$

Considering NOP, the *Component* and *Decorator* classes involve polymorphism. More specifically, they both have (m) virtual functions. Thus in system level,

$$NOP = \frac{(2 * m)}{3 + n + p + q}$$

The *Decorator* class inherits from the *Component* class, so its ANA equals 1. The number of ancestors for the (n) classes that represent $Leaf_i$ equals 1, for the (p) classes that represent $ConcreDecoratorA_i$ equals 1, and for the (q) classes that represent $ConcreDecoratorB_i$ equals 1. Thus,

$$ANA = \frac{1 + (1 * n) + (2 * p) + (2 * q)}{3 + n + p + q}$$

Furthermore, *Client* and *Decorator* have one private attribute (DAM=1). For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *Client* and *Component* classes hold (m) public methods. The *Decorator* class holds (m+2) public methods, the (n) $Leaf_i$ classes hold (m) public methods, the (p) $ConcreDecoratorA_i$ classes hold (2*m) public methods and the (q) $ConcreDecoratorB_i$ classes hold (m) public methods. Thus at system level,

$$CIS = \frac{(3 * m) + 2 + (m * n) + (2 * m * p) + (m * q)}{3 + n + p + q}$$

Finally, since the system does not contain any private or protected methods, the score of the NOM metric equals the score of the CIS metric. Thus,

$$NOM = \frac{(3 * m) + 2 + (m * n) + (2 * m * p) + (m * q)}{3 + n + p + q}$$

Alternative Literature Solution

The number of classes in the system is the sum of the number of $Leaf_i$ classes (n), plus 3 (*Decorator*, *Component* and *Client*). Thus,

$$DSC = 3 + n$$

The NOH in *Component* class equals 1, because it inherits from other classes, at the first level. The other classes do not inherit from any others, so their NOH equals 0. Thus,

$$NOH = 1$$

The *Decorator* and *Leaf* classes inherit all the methods from the *Component* class, so its MFA equals 0. For all the other classes, MFA is not defined. Thus,

$$MFA = 0$$

The *Client* class includes an object, of type Component, so its DCC equals 1. The *Component* class is abstract and does not reference any other object, so its DCC equals 0. The *Decorator* class inherits from the *Component* class and includes (n) objects, of type $Leaf_i$, so its DCC equals (n+1). Thus,

$$DCC = \frac{2 * n + 2}{3 + n}$$

The *Decorator* class has one parameter type to (n) sets of methods $addLeaf_i$ and $removeLeaf_i$ (CAM=$\frac{2}{2*n+m+p*m}$). Concerning the *Client, Component* and $Leaf_i$ classes, CAM is not defined. Thus,

$$CAM = \frac{2}{2 * n + m + p * m}$$

The *Client* class includes an object, of type *Component*, so its MOA equals 1. The *Decorator* class includes (n) objects, of type $Leaf_i$, so its MOA equals (n). All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

$$MOA = \frac{n + 1}{3 + n}$$

Considering NOP, the *Component* class involves polymorphism, so the *Component* class has (m) virtual functions. Thus in system level,

$$NOP = \frac{m}{3 + n}$$

The *Decorator* class inherits from the *Component* class, so its ANA equals 1. The number of ancestors for the (n) classes that represent $Leaf_i$ equals 1. For all the other classes, ANA equals 0.Thus,

$$ANA = \frac{1 + (1 * n)}{3 + n}$$

Furthermore, *Client* has one private attribute, so its DAM equals 1. The *Decorator* class has (n+1) private attributes, so its DAM equals 1. For all the other classes, its DAM is not defined. Thus,

$$DAM = 1$$

The *Client* and *Component* classes hold (m) public methods. The *Decorator* class holds (m+2*n+p*m) public methods and the (n) $Leaf_i$ classes hold (m) public methods. Thus at system level,

$$CIS = \frac{(3 * m) + (2 * n) + (p * m) + (m * n)}{3 + n}$$

Finally, since the system does not contain any private or protected methods, the score of the NOM metric equals the score of the CIS metric. Thus,

$$NOM = \frac{(3 * m) + (2 * n) + (p * m) + (m * n)}{3 + n}$$

3. Template Method

In this section we investigate the design quality of Template Method pattern. Template Method is used when "you want to define the skeleton of an algorithm in an operation, deferring some steps to client subclasses" [Gamma et al. 1995]. In section 3.1 we present the structure on the Template Method pattern and one alternatives design solution. In section 3.2 we present the results of applying the method.

3.1 Design Solutions

The class diagram of a typical Template Method instance is presented in Figure 3. The alternative design solution is presented in Figure 4.

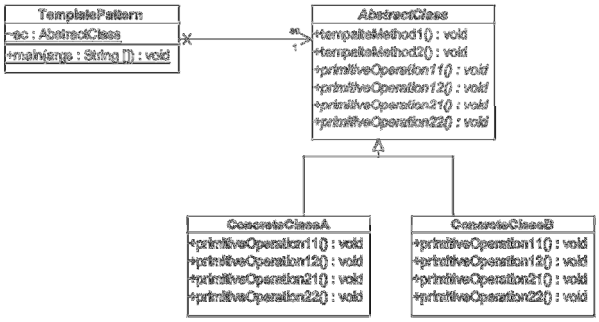

Figure 3. Template Method Design Pattern Class Diagram

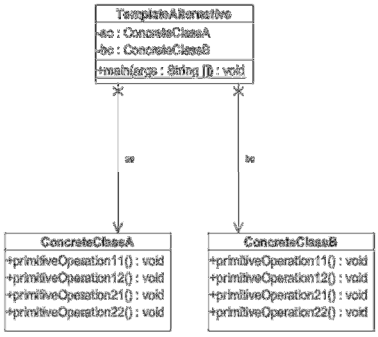

Figure 4. Template Method Design Alternative Class Diagram

We have identified three axes of change, based on one class hierarchy and two pattern-related methods.

Hierarchies:

- *Let n to be the number of Concrete Classes in the design.*

Methods:

- *Let the system have m template methods*
- *Let p to stand for the primitive operations used by the template methods*

The Template Alternative class holds direct references to every one of the (n) Concrete Classes and directly calls the set of methods that it desires. The notions (n), (m) and (p) are exactly the same as in the design pattern solution

3.2 Results

By taking into account the identified axes of change and the definition of the used metrics, we create the follow functions:

Pattern Solution

The number of classes in the system is the sum of the number of *ConcreteClass*$_i$ classes (n), plus 2 (*TemplatePattern* and *AbstractClass*). Thus,

$$DSC = 2 + n$$

The NOH in *AbstractClass* class equals 1 because it inherits from other classes, at the first level. The other classes do not inherit from any other, so their NOH equal 0. Thus,

$$NOH = 1$$

The (n) *ConcreteClass*$_i$ classes inherit only the primitiveOperation() method from the *AbstractClass* class, so its MFA equals $\left(\frac{m}{m+p}\right)$. For the other classes MFA equals 0. Thus,

$$MFA = \frac{n * m}{(m + p) * (n + 2)}$$

The *TemplatePattern* class includes one objects, of type *AbstractClass*, and creates objects of (n) *ConcreteClass*$_i$, so its DCC equals (n+1). The *AbstractClass* class is abstract and does not reference any other object, so its DCC equals 0. The (n) *ConcreteClass*$_i$ *classes* inherit from the *AbstractClass* class, so its DCC equals 1. Thus

$$DCC = \frac{1 + (2 * n)}{2 + n}$$

CAM cannot be defined for all system classes. Thus,

$$CAM = N/A$$

The *TemplatePattern* class includes an object of type AbstractClass, so its MOA equals 1. All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

$$MOA = \frac{1}{2+n}$$

Considering NOP, the *AbstractClass* involves polymorphism, so the *AbstractClass* has (p) virtual functions. Thus in system level,

$$NOP = \frac{p}{2+n}$$

The (n) $ConcreteClass_i$ classes inherit from the *AbstractClass* class, so its ANA equals 1. For all the other classes ANA equals 0. Thus,

$$ANA = \frac{(1*n)}{2+n}$$

Additionally, TemplatePattern class has one private attribute (DAM=1). For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *TemplatePattern* class holds one public method. The *AbstractClass* class holds (m+p) public methods and the (n) $ConcreteClass_i$ classes hold (p) public methods. Thus,

$$\mathrm{CIS} = \frac{(\mathrm{m}+\mathrm{p})+(\mathrm{p}*\mathrm{n})+1}{2+\mathrm{n}}$$

Finally, since the system does not contain any private or protected methods, the NOM metric equals CIS. Thus,

$$NOM = \frac{(m+p)+(p*n)+1}{2+n}$$

Alternative Solution

The number of classes in the system is the sum of the number of $ConcreteClass_i$ classes (n), plus 1 (*TemplateAlternative*). Thus,

$$DSC = 1+n$$

All the classes in the system do not present any hierarchy, so their NOH equal 0. Thus,

$$NOH = 0$$

MFA equals 0 for all system classes. Thus,

$$MFA = 0$$

The *TemplateAlternative* class includes (n) objects, of type ConcreteClass$_i$, so its DCC equals (n). The *ConcreteClass* class does not reference any other object, so its DCC equals 0. Thus,

$$DCC = \frac{n}{1+n}$$

CAM cannot be defined for all classes in the system. Thus,

$$CAM = N/A$$

The *TemplateAlternative* class includes (n) objects of type ConcreteClass$_i$, so its MOA equals 1. All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

$$MOA = \frac{n}{1+n}$$

The NOP metric for all classes equals 0, because there is no inheritance involved in the system.

$$NOP = 0$$

The ANA metric for all classes equals 0, because there is no inheritance involved in the system.

$$ANA = 0$$

Additionally, TemplateAlternative class has (n) private attributes (DAM=1). For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *TemplateAlternative* class holds one public method. The (n) *ConcreteClass$_i$* classes hold (m+p) public methods. Thus,

$$CIS = \frac{(n*(p+m))+1}{1+n}$$

Finally, since the system does not contain any private or protected methods, the NOM metric equals CIS. Thus,

$$NOM = \frac{(n*(p+m))+1}{1+n}$$

4. Strategy

In this section investigate the design quality of the Strategy design pattern. Strategy is used when "you want to alter the behavior of an algorithm at run-time" [Gamma et al. 1995]. In section 4.1 we present the structure on the Strategy pattern and one alternative design solution. In section 4.2 we present the results of methodology.

4.1 Design Solutions

The class diagram of a typical Strategy instance is presented in Figure 5. The alternative design solution is presented in Figure 6.

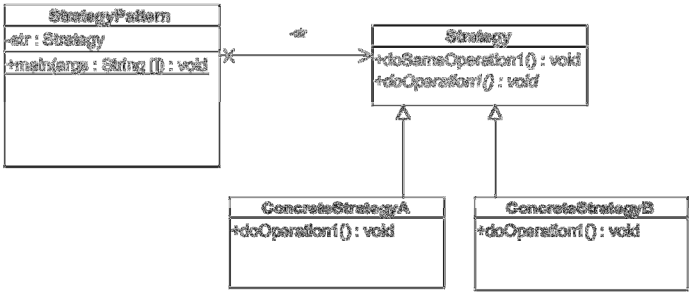

Figure 5. Strategy Design Pattern Class Diagram

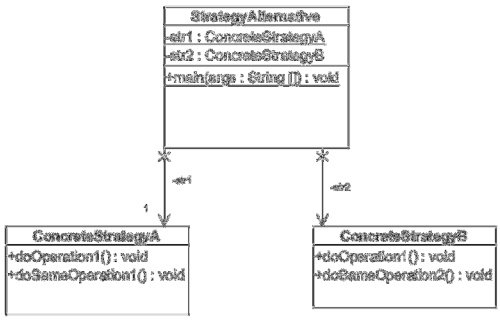

Figure 6. Strategy Design Alternative Class Diagram

We have identified three axes of change, based on the class hierarchies and two pattern-related methods.

Hierarchies:

- *Let n to be the number of Concrete Strategies.*

Methods:

- *Let m to be the number of operations, i.e. the number of abstract methods in the strategy class hierarchy*
- *Let q to be the number of methods that are inherited and not overridden in the hierarchy.*

The Strategy Alternative class holds references to Concrete Strategies. In addition to that the common behavior (q) methods of Concrete Strategies exists in both classes. It is intuitive that the higher the number of these methods, the higher the need for using the strategy design pattern. The notions of (n), (m) and (q) are equal to those of the design pattern solution.

4.2 Results

By taking into account the identified axes of change and the definition of the used metrics, we create the follow functions:

Pattern Solution

The number of classes in the system equals the sum of the number of *ConcreteStrategy*$_i$ classes (n), plus 2 (*StrategyPattern* and *Strategy*). Thus,

$$DSC = 2 + n$$

The NOH in *Strategy* class equals 1 because it inherits from other classes, at the first level. The other classes do not inherit from any other, so their NOH equal 0. Thus,

$$NOH = 1$$

The (n) *ConcreteStrategy*$_i$ classes inherit only the doOperation() methods from the *Strategy* class, so its MFA equals $\left(\frac{q}{m+q}\right)$. For the other classes MFA equals 0. Thus,

$$MFA = \frac{n * q}{(m+q) * (n+2)}$$

The *StrategyPattern* class includes one objects, of type *Strategy*, and creates objects of (n) *ConcreteStrategy*$_i$, so its DCC equals (n+1). The *Strategy* class is abstract and does not reference any other object, so its DCC equals 0. The (n) *ConcreteStrategy*$_i$ classes inherit from the *Strategy class*, so their DCC equals 1. Thus,

$$DCC = \frac{1 + (2 * n)}{2 + n}$$

For all classes in the system CAM cannot be defined. Thus,

$$CAM = N/A$$

The *StrategyPattern* class includes an object of type Strategy, so its MOA equals 1. All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

$$MOA = \frac{1}{2 + n}$$

Considering NOP, the *Strategy* class involves polymorphism, so the *Strategy* class has (m) virtual functions. Thus in system level,

$$NOP = \frac{m}{2 + n}$$

The (n) *ConcreteStrategy*$_i$ classes inherit the *Strategy* class, so their ANA equals 1. For all the other classes ANA equals 0. Thus,

$$ANA = \frac{n}{2+n}$$

Additionally, StrategyPattern class has one private variable (DAM=1). For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *StrategyPattern* class holds one public method. The *Strategy* class holds (m+q) public methods and the (n) *ConcreteStrategy*$_i$ classes hold (m) public methods. Thus,

$$CIS = \frac{(m+q)+(m*n)+1}{2+n}$$

Finally, since the system does not contain any private or protected methods, the NOM metric equals CIS. Thus,

$$NOM = \frac{(m*n)+(m+q)+1}{2+n}$$

Alternative Literature Solution

The number of classes in the system equals the sum of the number of *ConcreteStrategy*$_i$ classes (n), plus 1 (*StrategyAlternative*). Thus,

$$DSC = 1+n$$

All the classes in the system do not present any hierarchy, so NOH equals 0. Thus,

$$NOH = 0$$

For all classes in the system MFA equals 0. Thus,

$$MFA = 0$$

The *StrategyAlternative* class includes (n) objects, of type *ConcreteStrategy*$_i$, so its DCC equals (n). The *ConcreteStrategy*$_i$ classes do not reference any other object, so their DCC equal 0. Thus,

$$DCC = \frac{n}{1+n}$$

For all classes in the system CAM cannot be defined. Thus,

$$CAM = N/A$$

The *StrategyAlternative* class includes (n) objects, of type *ConcreteStrategy*$_i$, so its MOA equals (n). All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

$$MOA = \frac{n}{1+n}$$

The NOP metric for all classes equals 0, because there is no inheritance involved in the system.

$$NOP = 0$$

The ANA metric for all classes equals 0, because there is no inheritance involved in the system.

$$ANA = 0$$

Additionally, *StrategyAlternative* has (n) private variables (DAM=1). For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *StrategyAlternative* class holds one public method. The (n) $ConcreteStrategy_i$ classes hold (m+q) public methods. Thus,

$$CIS = \frac{(n * (q+m)) + 1}{1+n}$$

Finally, since the system does not contain any private or protected methods, the NOM metric equals CIS. Thus,

$$NOM = \frac{(n * (q+m)) + 1}{1+n}$$