Assessing the Impact of Design Patterns on Software Quality along Software Evolution: An Analytical Method

SOFIA CHARALAMPIDOU, University of Groningen, the Netherlands

SEREN SENCER, Sabanci University, Turkey

APOSTOLOS AMPATZOGLOU, University of Groningen, the Netherlands

ELVIRA MARIA ARVANITOU, University of Groningen, the Netherlands

PARIS AVGERIOU, University of Groningen, the Netherlands

IOANNIS STAMELOS, Aristotle University of Thessaloniki, Greece

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 paper's 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]. The properties and the metrics are listed below:

- "Design Size" - DSC (Design Size in Classes) metric. This metric is a count of the total number of classes in the design. Range of values $[0, +\infty)$
- "Hierarchies" - 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 $[0, +\infty)$
- "Encapsulation" - 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" - 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" - 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" - MOA (Measure of Aggregation) metric. This metric counts the number of data declarations whose types are user defined classes. Range of values [0, +∞)
- "Inheritance" - 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" - NOP (Number of Polymorphic Methods) metric. This metric counts the methods that can exhibit polymorphic behavior. Range of values [0, +∞)
- "Messaging" - CIS (Class Interface Size) metric. This metric is a count of the number of public methods in a class. Range of values [0, +∞)
- "Complexity" - 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 six axes of change, three based on the class hierarchies and three on 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.*
- *Let k be the number of otherOperation methods.*
- *Let r be the number of additionalOperation methods.*

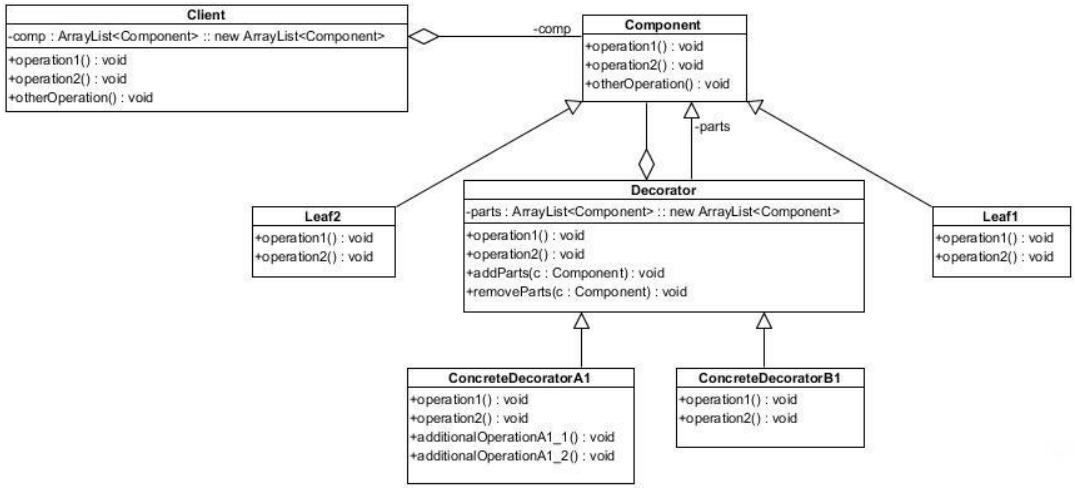

Figure 1. Decorator Design Pattern Class Diagram

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*.

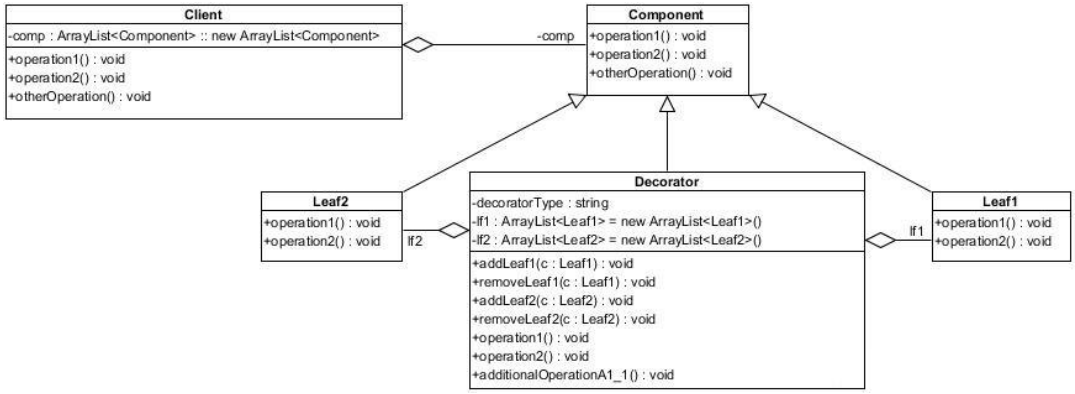

Figure 2. Decorator Design Alternative Class Diagram

2.2 Results

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

<u>Pattern Solution</u>

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 classes *Component* and *Decorator* have NOH = 1, because they inherit from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 1$$

The (p) *ConcreDecoratorA*$_i$ classes inherit two methods, i.e. addParts(c) and removeParts(c), from the *Decorator* class and (k) methods, i.e., otherOperation(), from the *Component* class. The (p) *ConcreDecoratorA*$_i$ classes have (m+r) local methods. Therefore, their MFA equals ($\frac{k+2}{k+r+m+2}$). The (q) *ConcreDecoratorB*$_i$ classes also inherit the same two methods from the *Decorator* class and the same (k) methods from the *Component* class, but they have (m) local methods. Therefore, their MFA equals($\frac{k+2}{k+m+2}$). The *Decorator* class inherits (k) methods, i.e. otherOperation(), from the *Component* class and it has (m+2) local methods, so its MFA equals ($\frac{k}{k+m+2}$). Thus,

$$\text{MFA} = \frac{\frac{p*(k+2)}{k+r+m+2} + \frac{q*(k+2)}{k+m+2} + \frac{k}{k+\ m+2}}{1+n+p+q}$$

The *Client* class includes an object, of type *Component*, so its DCC equals 1. The *Component* class includes an object, of type *Decorator*, 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 that is used in 2 methods. In total it holds (m+2) methods, thus CAM$_{Decorator}$ = ($\frac{2}{m+2}$) . For the other classes CAM is not defined.

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

The *Decorator* class includes an object of type *Component*, so $MOA_{Decorator} = 1$. The *Client* class includes an object of type *Component*, so $MOA_{Client} = 1$. 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 methods. 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 ($DAM_{Client} = 1$ and $DAM_{Decorator} = 1$). For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *Client* and *Component* classes hold (m+k) 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 (m+r) public methods and the (q) $ConcreDecoratorB_i$ classes hold (m) public methods. Thus at system level,

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

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,

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

<u>Alternative Literature Solution</u>

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 $NOH_{Component}$ = 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* class inherits (k) methods, i.e, otherOperation(), from the *Component* class and holds (2*n+m+r) local methods, so its MFA equals ($\frac{k}{k+2*n+r+m}$). The (n) *Leafi* classes also inherit the same (k) methods from the *Component* class and they have (m) methods, so their MFA equals($\frac{k}{k+m}$). Thus,

$$MFA = \frac{\frac{k}{k + 2 * n + r + m} + \frac{n * k}{k + m}}{1 + n}$$

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). The (n) $Leaf_i$ classes have inherit from the *Component* class, so their DCC equals 1. Thus,

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

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

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

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

$$CIS = \frac{(3 * m) + 2\mathrm{k} + (2 * n) + \mathrm{r} + (m * n)}{3 + n}$$

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,

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

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 alternative 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.

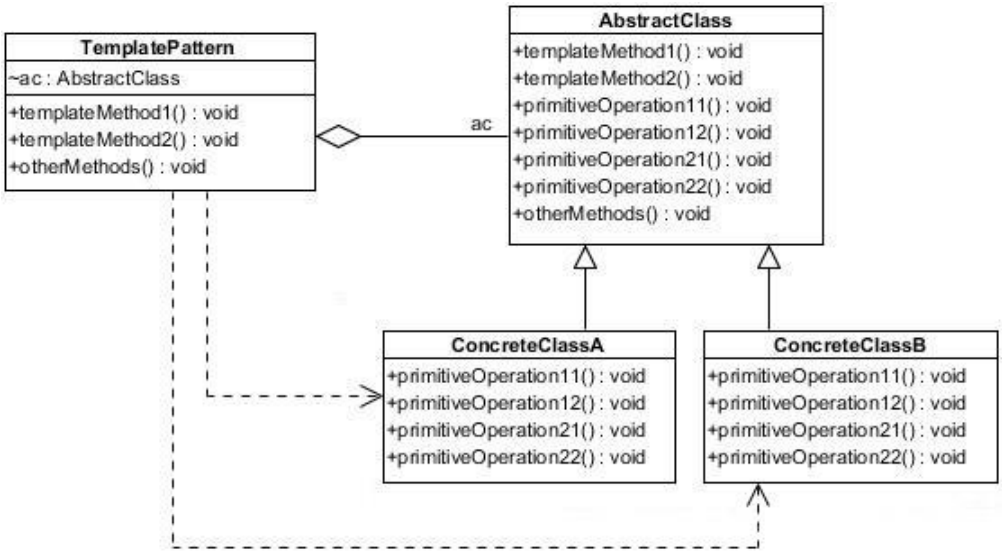

Figure 3. Template Method Design Pattern Class Diagram

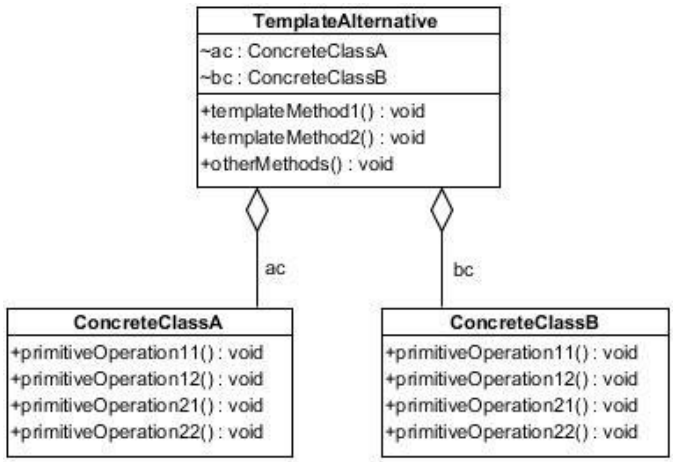

Figure 4. Template Method Design Alternative Class Diagram

We have identified four axes of change, one based on a class hierarchy and three 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*
- *Let the system have q other 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), (p) and (q) 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:

<u>Pattern Solution</u>

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 (m) templateMethod$_i$() methods and (q) otherMethods() from the *AbstractClass* class. The (n) *ConcreteClass*$_i$ classes have (p) methods, so their MFA equals$\left(\frac{m+q}{m+q+p}\right)$. For the other classes MFA is not defined. Thus,

$$MFA = \frac{m+q}{m+q+p}$$

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 MOA$_{Template}$ = 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 (m+q) public method. The *AbstractClass* class holds (m+q+p) public methods and the (n) *ConcreteClass*$_i$ classes hold (p) public methods. Thus,

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

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

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

<u>*Alternative Solution*</u>

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 (m+q) public method. The (n) *ConcreteClass$_i$* classes hold (p) public methods. Thus,

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

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

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

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.

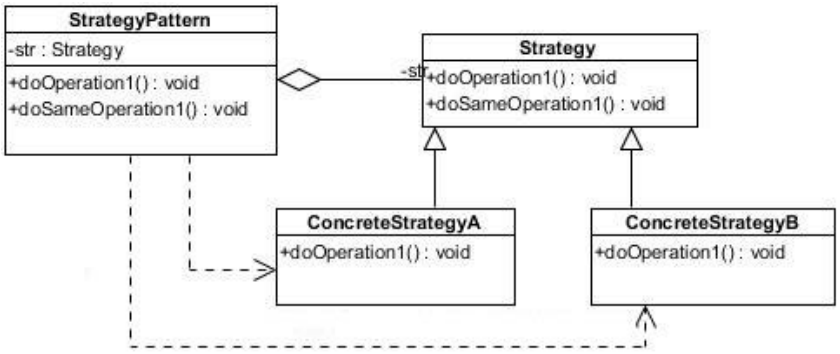

Figure 5. Strategy Design Pattern Class Diagram

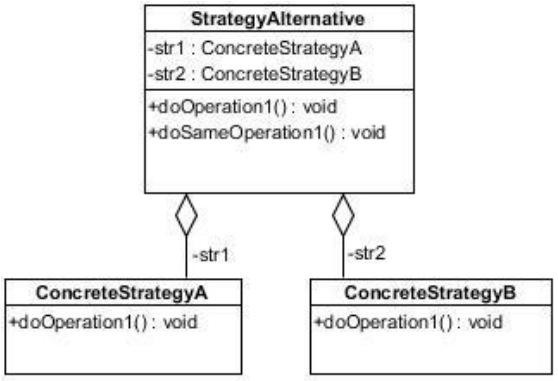

Figure 6. Strategy Design Alternative Class Diagram

We have identified three axes of change, one based on the class hierarchies and two based on 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 is not defined. Thus,

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

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

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

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

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

<u>*Alternative Literature Solution*</u>

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 (q+m) public method. The (n) *ConcreteStrategy*$_i$ classes hold (m+q) public methods. Thus,

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

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

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

5. Visitor

In this section investigate the design quality of the Visitor design pattern. Visitor is used when "you want to add new operations to existing object structures without modifying those structures" [Gamma et al. 1995]. In section 5.1 we present the structure on the Visitor pattern and one alternative design solution. In section 5.2 we present the results of methodology.

5.1 Design Solutions

The class diagram of a typical Visitor instance is presented in Figure 7. The alternative design solution is presented in Figure 8.

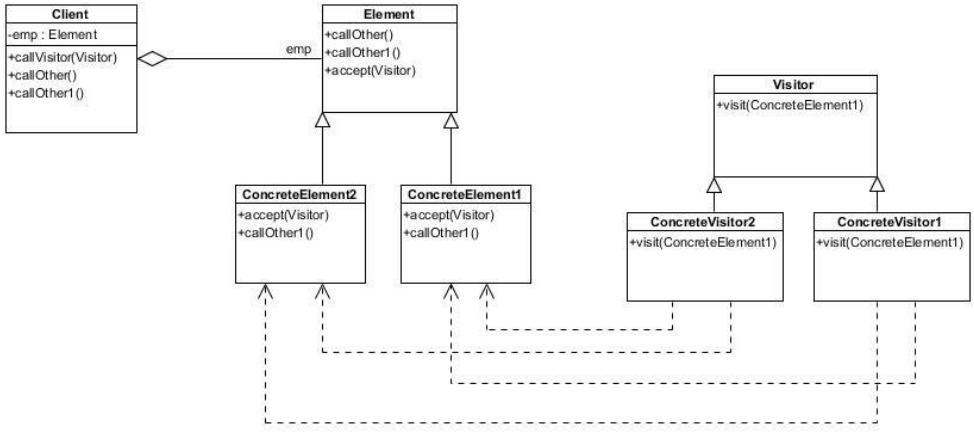

Figure 7. Visitor Design Pattern Class Diagram

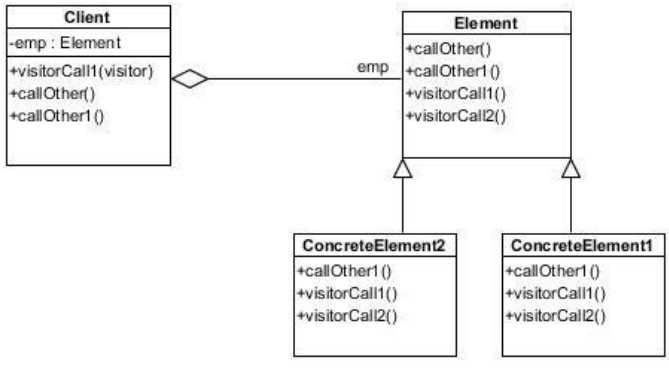

Figure 8. Visitor Design Alternative Class Diagram

We have identified 4 axes of change, two based on the class hierarchies and two based on pattern-related methods.

Hierarchies:

- *Let n to be the number of Concrete Elements.*
- *Let m to be the number of Concrete Visitors.*

Methods:

- *Let p to be the number of callOther methods that are the same for all ConcreteElements.*
- *Let q to be the number of callOther1 methods that are different for ConcreteElements and therefore are overridden in the hierarchy.*

In the alternative the Client class holds a reference to the elements hierarchy. However, since the visitor hierarchy is completely dropped, the operations performed through the visit methods are now located in the ConcreteElement classes. The notions of (n), (m), (p) and (q) are equal to those of the design pattern solution.

5.2 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 *ConcreteElement*$_i$ classes (n), plus number of *ConcreteVisitor*$_i$ classes (m), plus 3 (*Client*, *Element* and *Visitor*). Thus,

$$DSC = n + m + 3$$

The classes *Element* and *Visitor* have NOH = 1, because they inherit from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 2$$

The (n) *ConcreteElement*$_i$ classes inherit only the callOther() methods from the *Element* class, so their MFA equals $\left(\frac{p}{p+q+1}\right)$. The (m) *ConcreteVisitor*$_i$ classes inherit no methods from the *Visitor* class, so their MFA equals 0. For the other classes MFA is not defined. Thus,

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

The *Client* class includes one object of type *Element*, so its DCC equals 1. The *Element* and *Visitor* classes have DCC = 0. The (n) *ConcreteElement*$_i$ classes inherit from the *Element* class, so their DCC equals 1. The (m) *ConcreteVisitor*$_i$ classes inherit from the *Visitor* class and create objects of (n) *ConcreteElement*$_i$ classes, so their DCC equals (n+1). Thus,

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

The *Client* class has one type of parameter (Visitor) in (m) methods and (m+p+q) methods in total. So its CAM equals$\left(\frac{m}{m+p+q}\right)$. The *Element* class has one type of parameter (Visitor) and (1+p+q) methods. So its CAM equals$\left(\frac{1}{1+p+q}\right)$. The (n) *ConcreteElement*$_i$ classes have one argument and (q+1) methods so their CAM equals$\left(\frac{1}{1+q}\right)$. The *Visitor* and (m) *ConcreteVisitor*$_i$ classes have CAM =$\left(\frac{1}{n}\right)$. Thus,

$$CAM = \frac{\left(\frac{m}{m + p + q}\right) + \left(\frac{1}{1 + p + q}\right) + n * \left(\frac{1}{1 + q}\right) + (m + 1) * \left(\frac{1}{n}\right)}{m + n + 3}$$

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

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

Considering NOP, the *Element* class involves polymorphism, so the *Element* class has (q+1) virtual functions. The *Visitor* class involves polymorphism, so the *Visitor* class has (n) virtual functions. Thus in system level,

$$NOP = \frac{q+1+n}{m+n+3}$$

The (n) $ConcreteElement_i$ classes inherit the *Element* class, so their ANA equals 1. The (m) $ConcreteVisitor_i$ classes inherit the *Visitor* class, so their ANA equals 1. For all the other classes ANA equals 0. Thus,

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

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

$$DAM = 1$$

The *Client* class holds (p+q+m) public methods and the *Element* class holds (p+q+1) public methods. The (n) $ConcreteElement_i$ classes hold (q+1) methods. The *Visitor* class and the (m) $ConcreteVisitor_i$ classes hold (n) methods. Thus,

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

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

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

<u>*Alternative Literature Solution*</u>

The number of classes in the system equals the sum of the number of $ConcreteElement_i$ classes (n), plus 2 (*Client* and *Element*). Thus,

$$DSC = n+2$$

The *Element* class have NOH = 1, because it inherits from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 1$$

The (n) *ConcreteElement*$_i$ classes inherit (p) callOther() methods and they have (q+m) methods, so their MFA equals$\left(\frac{p}{p+q+m}\right)$. For the other classes MFA is not defined. Thus,

$$MFA = \frac{p}{p+q+m}$$

The *Client* class includes one object of type *Element*, so its DCC equals 1. The *Element* class has DCC = 0. The (n) *ConcreteElement*$_i$ classes inherit from the *Element* class, so their DCC equals 1. Thus,

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

The *Client* class has one type of parameter (Visitor) in (m) methods and (m+p+q) methods in total. So its CAM equals$\left(\frac{m}{m+p+q}\right)$. For the other classes, CAM is not defined because they have no attributes. Thus,

$$CAM = \frac{m}{m+p+q}$$

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

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

The *Element* class involves polymorphism, so the *Element* class has (m+q) virtual functions. The other classes have no virtual functions, so their NOP equals 0. Thus in system level,

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

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

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

The *Client* class has one private variable (DAM=1). For all the other classes have no attribute, so their DAM is not defined. Thus,

$$DAM = 1$$

The *Client* and *Element* classes hold (p+q+m) public methods. The (n) *ConcreteElement*$_i$ classes hold (q+m) methods. Thus,

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

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

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

6. Abstract Factory

In this section investigate the design quality of the Abstract Factory design pattern. Abstract Factory is used when "you want to separate the details of implementation of a set of objects from their general usage and rely on object composition, in the sense that object creation is implemented in methods exposed in the factory interface" [Gamma et al. 1995]. In section 6.1 we present the structure on the Abstract Factory pattern and one alternative design solution. In section 6.2 we present the results of methodology.

6.1 Design Solutions

The class diagram of a typical Abstract Factory instance is presented in Figure 9. The alternative design solution is presented in Figure 10.

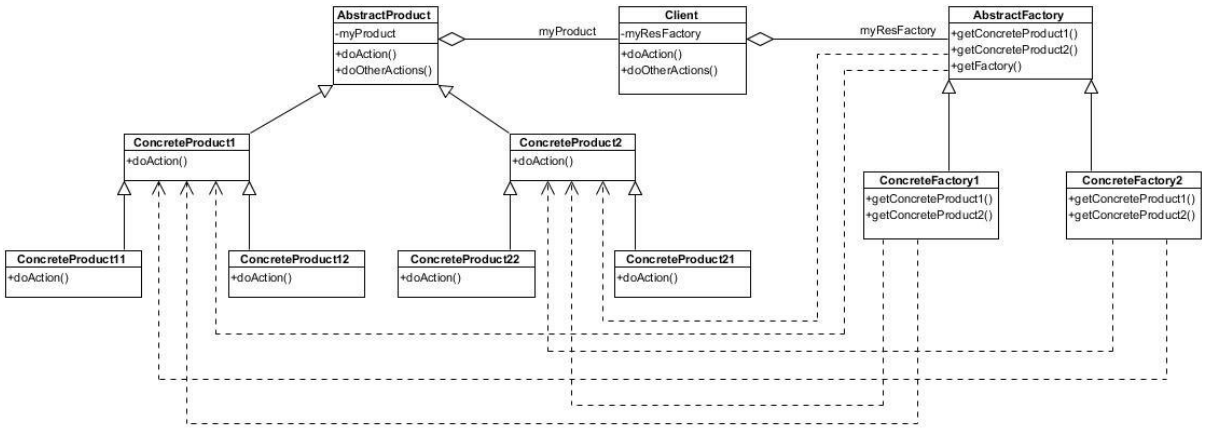

Figure 9. Abstract Factory Design Pattern Class Diagram

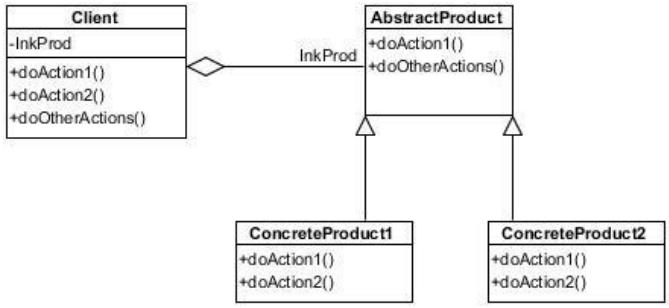

Figure 10. Abstract Factory Design Alternative Class Diagram

We have identified 3 axes of change, two based on the class hierarchies and one based on pattern-related methods.

Hierarchies:

- *Let n to be the number of Concrete Products.*
- *Let m to be the number of Concrete Factories.*

Methods:

- *Let p to be the number of doOtherActions methods.*

In the alternative solution, the Client class holds a reference to an Abstract Product. In addition to that the different behavior of each sub-family (m) is no evident in the doAction methods. The notions of (n), and (p) are equal to those of the design pattern solution.

6.2 Results

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

<u>*Pattern Solution*</u>

The number of classes in the system equals the sum of the number of $ConcreteProduct_i$ classes (n), plus number of $ConcreteProduct_{ij}$ classes (n*m), plus number of $ConcreteFactory_i$ classes (m), plus 3 (*AbstractProduct*, *Client* and *AbstractFactory*). Thus,

$$DSC = (n * m) + n + m + 3$$

The classes *AbstractProduct* and *AbstractFactory* have NOH = 1, because they inherit from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 2$$

The (n) $ConcreteProduct_i$ and the (n*m) $ConcreteProduct_{ij}$ classes inherit the doOtherActions() methods from the *AbstractProduct* class, so their MFA equals $\left(\frac{p}{p+1}\right)$. The (m) $ConcreteFactory_i$ classes inherit only the getFactory() method from the *AbstractFactory* class, so their MFA equals $\left(\frac{1}{n+1}\right)$. For the other classes MFA is not defined. Thus,

$$MFA = \frac{\left(\frac{p}{p+1}\right) * [(n * m) + n] + \left(\frac{1}{n+1}\right) * m}{(n * m) + n + m}$$

The *AbstractProduct* class includes one object of type *Client*, so its DCC equals 1. The *Client* class includes one object of type *AbstractFactory*, so its DCC equals 1. The (n) $ConcreteProduct_i$ classes

inherit from the *AbstractProduct* class, so their DCC equals 1. The (n*m) $ConcreteProduct_{ij}$ classes inherit from the $ConcreteProduct_i$ classes, so their DCC equals 1. The *AbstractFactory* class creates objects of (n) $ConcreteProduct_i$ classes, so its DCC equals (n). The (m) $ConcreteFactory_i$ classes inherit from the *AbstractFactory* class and create objects of (n) $ConcreteProduct_i$ classes, so their DCC equals (n+1). Thus,

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

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

$$CAM = N/A$$

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

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

The *AbstractProduct* class involves polymorphism, it has (1) virtual function. The (n) $ConcreteProduct_i$ classes have (1) virtual function. The *AbstractFactory* class involves polymorphism, it class has (n) virtual functions. Thus in system level,

$$NOP = \frac{1 + 2n}{(n * m) + n + m + 3}$$

The (n) $ConcreteProduct_i$ classes inherit from the *AbstractProduct* class, so their ANA equals 1. The (n*m) $ConcreteProduct_{ij}$ classes inherit from the $ConcreteProduct_i$ classes and the *AbstractProduct* class, so their ANA equals 2. The (m) $ConcreteFactory_i$ classes inherit from the *AbstractFactory* class, so their ANA equals 1. For all the other classes ANA equals 0. Thus,

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

The *AbstractProduct* and *Client* classes have one private variable, so their DAM equals 1. For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *AbstractProduct* and *Client* classes hold (p+1) public methods. The (n) $ConcreteProduct_i$ and (n*m) $ConcreteProduct_{ij}$ classes hold (1) method. The *AbstractFactory* class holds (m+1) public methods. The (m) $ConcreteFactory_i$ classes hold (n) public methods. Thus,

$$CIS = \frac{2p + 2 + n + (n * m) + m + 1 + (n * m)}{(n * m) + n + m + 3}$$

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

$$NOM = \frac{2p + 2 + n + (n * m) + m + 1 + (n * m)}{(n * m) + n + m + 3}$$

Alternative Literature Solution

The number of classes in the system equals the sum of the number of *ConcreteProduct*$_i$ classes (n), plus 2 (*Client* and *AbstractProduct*). Thus,

$$DSC = n + 2$$

The *AbstractProduct* class have NOH = 1, because it inherits from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 1$$

The (n) *ConcreteProduct*$_i$ classes inherit (p) doOtherActions() methods and they have (m) methods, so their MFA equals$\left(\frac{p}{p+m}\right)$. For the other classes MFA is not defined. Thus,

$$MFA = \frac{p}{p + m}$$

The *Client* class includes one object of type *AbstractProduct*, so its DCC equals 1. The *AbstractProduct* class has DCC = 0. The (n) *ConcreteProduct*$_i$ classes inherit from the *AbstractProduct* class, so their DCC equals 1. Thus,

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

For all classes in the system CAM cannot be defined because they have no parameters. Thus,

$$CAM = N/A$$

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

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

The *Element* class involves polymorphism, so the *Element* class has (m+q) virtual functions. The other classes have no virtual functions, so their NOP equals 0. Thus in system level,

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

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

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

The *Client* class has one private variable (DAM=1). For all the other classes have no attribute, so their DAM is not defined. Thus,

$$DAM = 1$$

The *Client* class and the *AbstractProduct* class hold (p+m) public methods. The (n) *ConcreteProduct*$_i$ classes hold (m) methods. Thus,

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

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

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

7. Bridge

In this section investigate the design quality of the Bridge design pattern. Bridge is used when "you want to decouple an abstraction from its implementation so that the two can vary independently" [Gamma et al. 1995]. In section 7.1 we present the structure on the Bridge pattern and one alternative design solution. In section 7.2 we present the results of methodology.

7.1 Design Solutions

The class diagram of a typical Bridge instance is presented in Figure 11. The alternative design solution is presented in Figure 12.

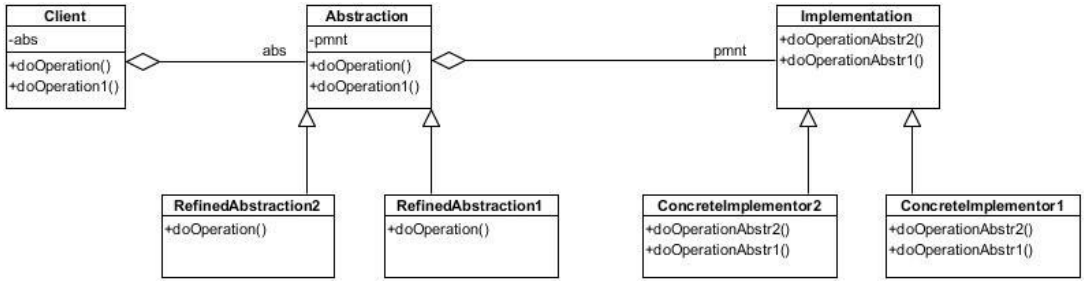

Figure 11. Bridge Design Pattern Class Diagram

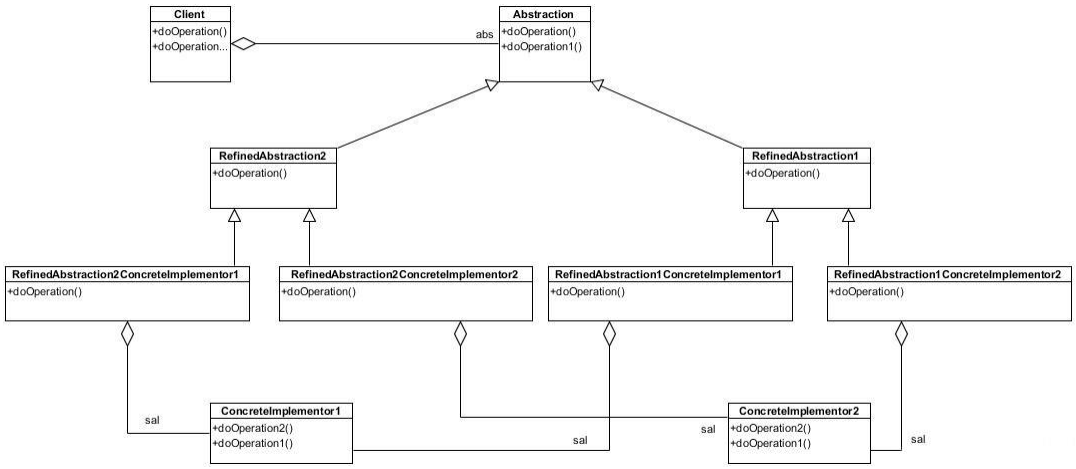

Figure 12. Bridge Design Alternative Class Diagram

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

Hierarchies:

- *Let n to be the number of Refined Abstractions.*
- *Let m to be the number of Concrete Implementors.*

Methods:

- *Let p to be the number of doOperation methods.*
- *Let q to be the number of doOperation1 methods.*

In the Bridge alternative a higher depth of inheritance has been preferred over class composition. Specifically, a combination of every Refined Abstraction and Concrete Implementors is placed on the third level of the Abstraction hierarchy. The notions of (n), (m), (p) and (q) are the same as those of the design pattern solution.

7.2 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 $RefinedAbstraction_i$ classes (n), plus number of $ConcreteImplementor_i$ classes (m), plus 3 (*Client*, *Abstraction* and *Implementation*). Thus,

$$DSC = n + m + 3$$

The classes *Abstraction* and *Implementation* have NOH = 1, because they inherit from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 2$$

The (n) *RefinedAbstraction*$_i$ classes inherit the (q) doOperation1() methods from the *Abstraction* class, so their MFA equals $\left(\frac{q}{p+q}\right)$. The (m) *ConcreteImplementor*$_i$ classes inherit no methods from the *Implementation* class, so their MFA equals 0. For the other classes MFA is not defined. Thus,

$$MFA = \frac{\left(\frac{q}{p+q}\right) * n}{n+m}$$

The *Client* class includes one object of type *Abstraction*, so its DCC equals 1. The *Abstraction* class includes one object of type *Implementation*, so its DCC equals 1. The (n) *RefinedAbstraction*$_i$ classes inherit from the *Abstraction* class, so their DCC equals 1. The (m) *ConcreteImplementor*$_i$ classes inherit from the *Implementation* class, so their DCC equals 1. Thus,

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

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

$$CAM = N/A$$

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

$$MOA = \frac{2}{n+m+3}$$

The *Abstraction* class involves polymorphism, it has (p) virtual function. The *Implementation* class have (n) virtual function. Thus in system level,

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

The (n) *RefinedAbstraction*$_i$ classes inherit from the *Abstraction* class, so their ANA equals 1. The (m) *ConcreteImplementor*$_i$ classes inherit from the *Implementation* class, so their ANA equals 1. For all the other classes ANA equals 0. Thus,

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

The *Client* and *Abstraction* classes have one private variable, so their DAM equals 1. For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *Client* and *Abstraction* classes hold (p+q) public methods. The (n) *RefinedAbstraction*$_i$ classes hold (p) method. The *Implementation* class and the (m) *ConcreteImplementor*$_i$ classes hold (n) public methods. Thus,

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

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

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

Alternative Literature Solution

The number of classes in the system equals the sum of the number of *RefinedAbstraction*$_i$ classes (n), plus number of *ConcreteImplementor*$_i$ classes (m), plus number of *RefinedAbstraction*$_i$*ConcreteImplementor*$_j$ classes (n*m), plus 2 (*Client* and *Abstraction*). Thus,

$$DSC = (n * m) + n + m + 2$$

The classes *Abstraction* have NOH = 1, because they inherit from other classes, at the first level. The other classes do not inherit from any others, so they have NOH = 0. The NOH metric is calculated in system level, thus

$$NOH = 1$$

The (n) *RefinedAbstraction*$_i$ classes and the (n*m) *RefinedAbstraction*$_i$*ConcreteImplementor*$_j$ classes inherit the (q) doOperation1() methods from the *Abstraction* class, so their MFA equals $\left(\frac{q}{p+q}\right)$. For the other classes MFA is not defined. Thus,

$$MFA = \frac{\left(\frac{q}{p + q}\right) * (n + (n * m))}{n + (n * m)} = \left(\frac{q}{p + q}\right)$$

The *Client* class includes one object of type *Abstraction*, so its DCC equals 1. The *Abstraction* class has DCC=0. The (n) *RefinedAbstraction*$_i$ classes inherit from the *Abstraction* class, so their DCC equals 1. The (n*m) *RefinedAbstraction*$_i$*ConcreteImplementor*$_j$ classes inherit from the *Abstraction* class and include one object of type *ConcreteImplementor*$_i$, so their DCC equals 2. The (m) *ConcreteImplementor*$_i$ classes have DCC=0. Thus,

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

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

$$CAM = N/A$$

The *Client* class includes one object of type *Abstraction*, so its MOA equals 1. The (n*m) *RefinedAbstraction*$_i$*ConcreteImplementor*$_j$ classes include one object of type *ConcreteImplementor*$_i$, so their MOA equals 1. All other classes do not include any aggregations or compositions to other classes, so their MOA equals 0. Thus,

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

The *Abstraction* class involves polymorphism, it has (p) virtual methods. The (n) *RefinedAbstraction*$_i$ classes have (p) virtual methods too. Thus in system level,

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

The (n) *RefinedAbstraction*$_i$ classes inherit from the *Abstraction* class, so their ANA equals 1. The (n*m) *RefinedAbstraction*$_i$*ConcreteImplementor*$_j$ classes inherit from the *Abstraction* class and *RefinedAbstraction*$_i$ classes, so their ANA equals 2. For all the other classes ANA equals 0. Thus,

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

The *Client* class has one private variable, so its DAM equals 1. For all the other classes, DAM is not defined. Thus,

$$DAM = 1$$

The *Client* and *Abstraction* classes hold (p+q) public methods. The (n) *RefinedAbstraction*$_i$ classes and the (n*m) *RefinedAbstraction*$_i$*ConcreteImplementor*$_j$ hold (p) public methods. The (m) *ConcreteImplementor*$_i$ classes hold (p*n) public methods. Thus,

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

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

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