Based on:_Charalampidou, S. Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P., Sencer, S., Arvanitou, E. M., and Stamelos, I. (20017b). A Theoretical Model for Capturing the Impact of Design Patterns on Quality: The Decorator Case Study. In 32nd ACM Symposium on Applied Computing (SAC), ACM, New York, NY, USA, 1231-1238.

5 A Theoretical Model for Capturing the Impact of Design Patterns on Quality: The Decorator Case Study

Abstract

Design patterns are widely recognized as reusable solutions that can be applied for improving design quality. However, empirical results suggest that patterns may sometimes support and other times hurt a quality attribute. Thus, there is a need for guidance on when a pattern is beneficial and when it is not. To provide such guidance, we propose a theoretical model for understanding the effect of patterns on quality. The obtained results are expected to improve the theoretical body of knowledge on design patterns, and facilitate informed decision making about when to insert or remove a pattern from a system. As an example, we present and discuss the results of modelling and exploring the effect of Decorator instances on quality. The results suggest that Decorator instances that are not expected to evolve through the addition of components in composite objects decrease system cohesion and therefore, modularity and maintainability are weakened.

5.1 Introduction

GoF design patterns (Gamma et al. 1995) are widely adopted as reusable solutions to common design problems. Although these patterns were not originally explicitly linked to quality attributes, a recent systematic secondary study (Ampatzoglou et al. 2013b) identified an extensive corpus of research aiming at assessing the effect of GoF patterns on quality. However, the study indicates that GoF patterns cannot be *uniformly evaluated with respect to their effect on quality*; specifically, different empirical studies suggest that the same pattern exhibits exactly the opposite effect on the same quality attribute (Ampatzoglou et al. 2013b), e.g., Visitor has been evaluated both as positively (Khomh and Guéhéneuc 2008) and as negatively (e.g., (Vokác et al. 2004)) related to understandability.

To investigate the aforementioned contradictory results, a few theoretical approaches have been proposed. These approaches (see Section 5.2) develop mathematical models that capture the effect of patterns on quality, by considering the generic representation of a pattern, rather than a specific instance. Such approaches have modeled the effect of patterns on various qualities. Nevertheless, only limited patterns have been explored, while the effect of patterns is mostly studied on the directly affected quality attributes (e.g., the effect of patterns involving polymorphism on the number of polymorphic methods). Therefore, we need to explore additional patterns, and model their effect on a variety of qualities, so as to identify possible trade-offs, i.e., positive effect on one quality attribute and negative effect to others.

The goal of this chapter is to thoroughly investigate the effect of the Decorator pattern on various qualities and study the corresponding trade-offs. To achieve this goal, we reuse and extend a two-step method (Ampatzoglou et al. 2012): (a) we first develop a theoretical model that captures the effect of patterns on quality attributes, based on numerical indicators, and (b) we then simulate all possible pattern instances based on the aforementioned model, in order to explore changes in the effect of patterns on quality. Specifically, during the second step of the method we perform statistical analysis to explore how frequently the pattern has a positive effect on quality. For patterns that do not have a uniform effect, we 'dig deeper' to identify the parameters that constitute the pattern beneficial or harmful. To demonstrate the method in this manuscript, we compare Decorator to a specific design alternative, and report the results. In an accompanying technical report[29] we present results on the State/Strategy and the Template Method patterns.

The rest of the chapter is organized as follows: in Section 5.2, we present *related work*; in Section 5.3, we introduce our *method* for comparing patterns to alternative solutions. In Section 5.4 we present the *application of* the method on *Decorator*, while in Section 5.4.5 the obtained *results*. In Section 5.5, we *discuss the findings* and present *implications* for *researchers and practitioners*. Section 5.7 outlines *threats to validity*, and Section 5.8 *concludes* this chapter.

5.2 Related Work

As related work we have considered studies that investigate *the effect of patterns on quality through theoretical models*. First, Huston (2001) studied the effect of three patterns (Mediator, Bridge and Visitor) on coupling, inheritance and size metrics. According to Huston, there are several thresholds that, when surpassed, the pattern application is beneficial. The differences between our work and (Huston 2001) are that: (a) we explore more qualities, quantified by different metrics, and (b) we investigate different patterns.

The second study by Hsueh et al. (2008) investigated the effect of six patterns on a single quality attribute that each pattern directly affects (e.g. the effect of: Observer on Coupling, Strategy on Polymorphism etc.). However, investigating the effect of a pattern on a single quality attribute can result in neglecting possible trade-offs that pattern usage induces. For example, when a pattern is employed, the coupling of the system may decrease, but as a side effect the size may increase. Compared to Hsueh et al. (2008), we do not limit our study to a single quality attribute, but we examine all metrics of the selected metric suite. Although both works examine the Decorator pattern, we advance the state of knowledge by studying 10 additional quality attributes.

[29] www.cs.rug.nl/search/uploads/Resources/patterns_TR_20151015.pdf

Finally, in (Ampatzoglou et al. 2012) the authors used the methodology proposed by Huston (2001) and Hsueh et al. (2008), to perform a comprehensive evaluation on the effect of three patterns (Bridge, Abstract Factory, and Visitor) on various qualities. The results of that study validated the existence of thresholds (named as "cut-off points") that when surpassed pattern application becomes beneficial. Our current work is based on this work, i.e., (Ampatzoglou et al. 2012), investigating a new pattern and using additional pattern-related characteristics.

5.3 Method

In this section we describe the method we applied for the needs of our study. The method is an enhanced version of the one introduced by Ampatzoglou et al. (2012). Our method is based on three fundamental observations, made on GoF patterns:

- Existence of a set of comparable solutions*:* For each pattern, we can propose several alternative design solutions (i.e. pattern and non-pattern) that can substitute its functionality and can be used in cases when the pattern is not beneficial (Ampatzoglou et al. 2013c).

- Existence of characteristics related to software quality attributes*:* GoF patterns contain certain structural characteristics that are related to quality. For example, in Bridge, such characteristics (i.e., class hierarchies, polymorphic method behavior, and class composition) improve maintainability and flexibility of the design (Ampatzoglou and Chatzigeorgiou 2007). Hence, measures on pattern-related characteristics, which evolve during maintenance, such as the number of polymorphic methods, or the classes in a hierarchy, can be used as parameters to our method to predict the effect on these qualities.

- Different instances of patterns vary with respect to the previously mentioned characteristics*:* Depending on how the patterns are instantiated in a particular system, measures of their structural characteristics may differ substantially, e.g., number of participating classes (Ampatzoglou et al. 2011). This might be an objective factor for the varying effect of different pattern instances on the same quality attribute.

Based on these observations, we developed a method consisting of two parts. In the first part (***model construction***) we derive equations that calculate quality metric scores for different pattern instances as a function of pattern-related characteristics.

In the second part (***analytical exploration***) we use statistical analysis on these models, to compare pattern and alternative design solutions.

Part A – Model Construction

1) *Identification of Alternatives*: Derive one or more alternative design solutions from literature, open-source solutions or designers' personal experiences.
2) *Identification of Pattern-related Parameters*: Identify the major modification operations, with respect to structural characteristics (i.e., add classes in hierarchies, or add pattern-related methods). Based on the modification operations that can be applied on the pattern, extract a list of pattern-related parameters (numerical indicators) that can characterize a specific instance. For example, in the Template Method, there is one parameter related to the number of concrete classes (altered through the *add concrete subclasses* modification operation) and two parameters related to methods: the number of template methods (*add inherited methods*) and the number of primitive operations (*add overridden methods*).
3) *Modeling of Solutions*: Model the alternative solutions identified in step 1, based on all the involved parameters of step 2.
4) *Quality model selection*: Select a quality model that fits the designer's needs, or simply a set of metrics. Any development team can select if they want to evaluate their solutions with respect to an existing quality model, or a customized model, or just a set of metrics that are not aggregated or composed.
5) *Construction of equations*: Construct equations that calculate quality attributes/metric scores as functions of pattern-related parameters (see step 2 and step 3).

Part B – Analytical Exploration

6) *Statistical Analysis*: Substitute the variables of the equations with the values that the pattern-related parameters are expected to get along pattern evolution. Perform descriptive statistics and hypothesis testing on the dataset.
7) *Cut-off Points Analysis*: If the results of the statistical analysis do not indicate which design solution is better, compare the equations of step 5 and identify the cut-off points (i.e., the solutions of the inequalities). The identified cut-off points suggest the values of pattern-related parameters for which each design solution (pattern or alternative) is beneficial.

The major difference between this method, compared to the original one (Ampatzoglou et al. 2012), which considered only one **class-related parameter**[30] (i.e., *number of concrete subclasses*) (Ng et al. 2007), lies on the identification of additional parameters[31] (i.e., *number of pattern-related methods*). Especially for the case of Decorator, studying pattern-related methods is important, since according to Di Penta et al. (2008) adding and removing methods is the most frequently applied modification operation. Instead of using the number of pattern-related methods, we decided to use more fine-grained parameters, based on the type of the method: (a) ***number of abstract methods, (b) number of overridden methods***, and (c) ***number of inherited methods***. The rationale of this decision is based on the fact that for some patterns (e.g., Strategy) the basic criterion for applying them can be the number of inherited methods compared to the number of overridden ones. In particular, if the number of overridden methods (varying behavior) is negligible compared to inherited methods (common behavior), then an alternative design (e.g., set of if-statements) might be preferable.

5.4 Model Construction

This section presents the application of the proposed method on Decorator, organized based on the first five steps of the method that correspond to the model construction. The last two steps (analytical exploration) are presented in Section 5.5.

5.4.1 Identify alternatives

Decorator is used for "adding behavior or state to individual objects at run-time" (Gamma et al. 1995). We selected to demonstrate the method on Decorator, due to its inherent complex structure and the fact that method-related parameters cannot

[30] We have not considered the "add clients" (Ng et al. 2007) parameter due to its uniform effect on both solutions. "Adding abstract classes" (Ng et al. 2007) was not considered, since the addition of an abstract class in a pattern instance would create a coupled pattern.

[31] This does not imply that the results of (Ampatzoglou et al. 2012) are invalidated, since for all examined patterns in (Ampatzoglou et al. 2012), the number of pattern-related methods is associated to the number of concrete subclasses.

be subsumed by the number of classes. The class diagram of a typical *Decorator* is presented in Figure 5.1, whereas an alternative design is presented in Figure 5.2. While building the alternative, we replaced: (***a***) the composition to objects of the superclass (i.e., link between Component and Decorator) with direct compositions to all leafs (i.e., link between Leaf1 and Decorator, etc.), which can be considered as a common design decision from novice software developers; and (***b***) the use of polymorphism (Decorator hierarchy) with conditional statements, based on the value of the decoratorType variable. Similarly, this is a common design decision—see refactoring: "prefer conditional over polymorphism" (Fowler et al. 1999).

We note that the specific alternative is not a pattern variant, but an artificial design constructed by ignoring some pattern principles. We acknowledge that the results reported in this chapter depend on this alternative, and would be different if we used a different alternative (see threats to validity in Section 5.7). In any case, one who wishes to apply the proposed method with another design alternative can reproduce the steps of the method, as illustrated in Section 5.4 to compare any set of design options.

5.4.2 Identify pattern-related parameters

In the structure of the Decorator design pattern we have identified ***six pattern-related parameters*** (see Figure 5.1): three based on the class hierarchies and three based on methods.

Number of Classes

- Let **n** be the number of Leaf$_i$ in the design.
- Let **p** be the number of ConcreteDecoratorA1$_i$—those that provide additional methods than the ones provided by the given methods of the hierarchy.
- Let **q** be the number of ConcreteDecoratorA2$_i$—those that only exhibit different behavior on the given methods of the hierarchy, without providing new ones.

Number of Methods

- Let **m** be the number of operation$_i$ methods—abstract methods in the Decorator class hierarchy.

- Let **k** be the number of otherOperation methods—non-abstract (inherited) methods in the Component class.
- Let **r** be the number of additionalOperation methods, offered by ConcreteDecoratorA1$_i$ classes.

In Figure 5.1 we demonstrate how the specific pattern-related parameters are mapped to the Decorator UML class diagram.

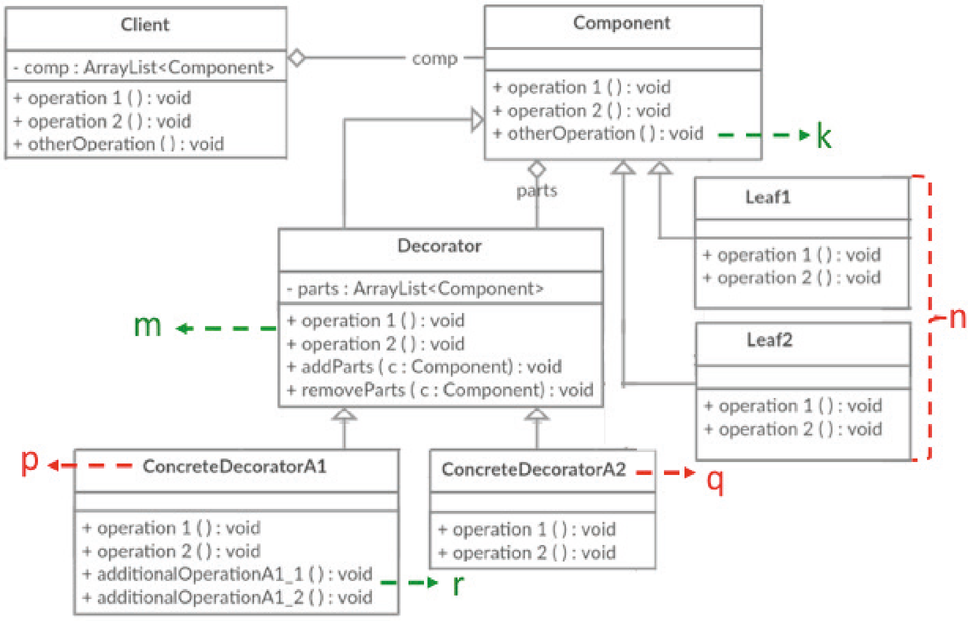


Figure 5.1: Decorator Design Pattern Class Diagram

5.4.3 Model solutions based on parameters

As explained in Section 5.4.1 the *Decorator Design Alternative* holds different 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 takes (p + q) possible values. Inside the (**m**) operation, we placed (**p+q**) if statements, to handle all possible ConcreteDecorator$_i$ classes. The way that the pattern-related parameters are mapped into the alternative UML class diagram is depicted in Figure 5.2. We note that (p) and (q) are not represented, since if-statements are not visible at class diagrams.

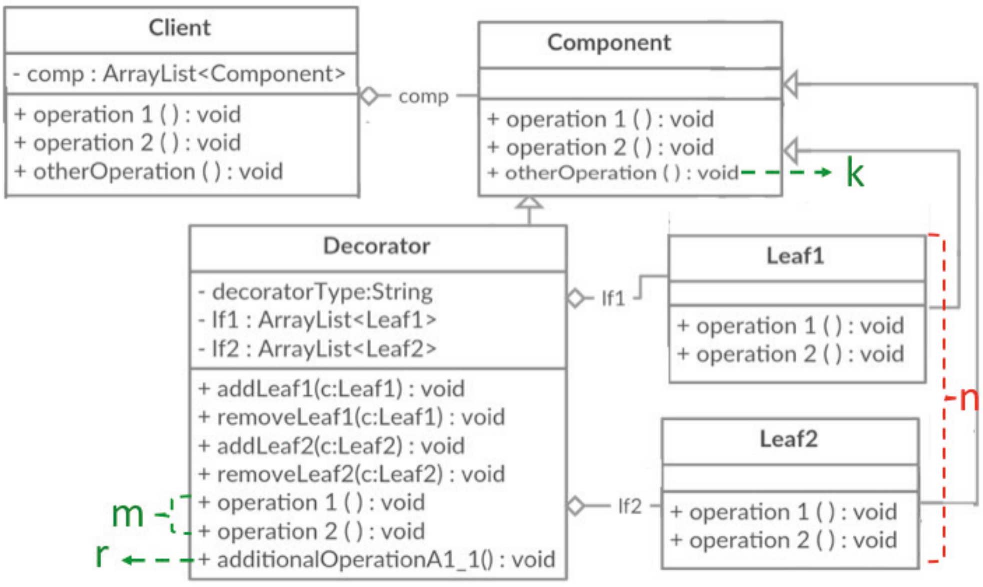


Figure 5.2: Decorator Design Alternative Class Diagram

5.4.4 Select a metric suite

For this study, we used the QMOOD metrics (Bansiya and Davis 2002). These metrics can directly quantify a set of low-level Quality Attributes (QA)—e.g., coupling, cohesion, etc., which in turn can be grouped to assess high-level ones (e.g., reusability, etc.). These low-level qualities and the metrics that quantify them are presented in Table 5.1 (Bansiya and Davis 2002). We note that in this study we use only the QMOOD metric definitions and their positive/negative relationship to high level quality attributes, rather than the mathematical formulas that are suggested for their quantification, so as not to raise a threat to construct validity (Hsueh et al. 2008) (see Section 5.7).

Table 5.1: QMOOD Metrics and Low-Level Quality Attributes

Low-Level QA	Metric Description
Design Size	**Design Size in Classes (DSC)** - Count of classes.
Messaging	**Class Interface Size (CIS)** - Count of public methods
Polymorhism	**Number of Polymorphic Methods (NOP)** - Number of methods that can exhibit polymorphic behavior
Abstraction	**Average Number of Ancestors (ANA)** - Average number of classes from which a class inherits.
Encapsulation	**Data Access Metric (DAM)** - Ratio of the number of private/protected fields to the total number of fields
Coupling	**Direct Class Coupling (DCC)** - Number of other classes that the class is directly related to.
Composition	**Measure of Aggregation (MOA)** - Number of data declarations whose types are user defined classes.
Inheritance	**Measure of Functional Abstraction (MFA)** - Ratio of number of methods inherited by total number of accessible methods.
Cohesion	**Cohesion Among Methods (CAMC)** - Sum of the intersection of a method parameters with the maximum set of all parameter types in the class.
Hierarcies	**Number of Hierarchies (NOH)** - Count of hierarchies in the design.
Complexity	**Number of Methods (NOM)** - Number of methods in the class.

5.4.5 Construct equations

By calculating the metric presented in Table 5.1 on the designs of Section 5.4.1, we formulated the metric scores for low-level quality attributes, for both solutions (f(x) for the pattern and g(x) for the alternative). The calculations are reported together with the obtained results for two additional patterns (i.e., Strategy and Template Method), in an accompanying technical report[1], due to space limitations. However, to enhance the readers' understandability, we provide the calculation of one metric (DCC) for the pattern (Decorator) solution, as an example. We clarify that to aggregate metric scores from the class level to the pattern level we use the average function. More specifically the numerator is calculated as the sum of the DCC of all classes, whereas the denominator equals the number of classes.

According to the class diagram presented in Figure 5.1, for the pattern solution, the numerator is calculated as follows: The Client class includes an object, of type Component, so its DCC equals 1. Similarly, the Component class includes an object, of type Decorator, so its DCC also equals 1. The (**n**) Leaf$_i$ classes inherit from the Component class, so their DCC equals 1. Similarly, the (**p**) ConcreDecoratorA1$_i$ classes and the (**q**) ConcreDecoratorA2$_i$ inherit from the Decorator class, so their DCC equals 1. The DCC of the Decorator class equals 0 since it does not include any dependencies. The denominator on the other hand, as already mentioned above is the number of classes in the pattern solution, i.e., the sum of the number of Leaf$_i$ classes (**n**), the number of ConcreDecoratorA1$_i$ classes (**p**), the number of ConcreDecoratorA2$_i$ classes (**q**), plus 3 (i.e. Decorator, Component and Client). Thus,

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

Similarly, we calculate the metric for the alternative solution, by considering the classes and methods of the respective design.

5.5 Analytical Results

In this section we present the results obtained while applying the second part of our method, in which we analyze the theoretical models constructed in Section 5.4. In Section 5.5.1 we present the results of the performed statistical analysis, so as to present quality attributes for which the pattern or the alternative solution is always beneficial (step 6). In Section 5.5.2 we explore the cases that no optimal design solution could be identified, by investigating the range values of pattern-related parameters for which each design solution is beneficial (step 7).

5.5.1 Statistical Analysis

In this section we present the results of our study obtained by substituting the variables of the equations with the most common values of pattern-related parameters, according to the literature[32]. In particular, based on a case study performed by Ampatzoglou et al. (2011) on 108 open source projects, Decorator instances tend to have on average 13 classes. Additionally, regarding the method-related parameters, literature suggests that classes (regardless of their pattern participation) rarely have more than 15 methods (Kalpana 2011). Based on the aforementioned claims, we can assume that:

- $n + p + q + 3 = 13$
- $n, q, p \in [1, 8]$
- $m, k, r \in [1, 13]$
- $\max (m + k + r) \leq 15$

By using the aforementioned rules as a way to obtain a sample that represents the most frequently occurring pattern instances, we developed a dataset consisting of 16,500 cases. By exploring this dataset using statistical analysis we aim at identifying the existence of differences between the two solutions in the most common design pattern occurrences.

In Table 5.2 each row represents one low-level quality attribute, whereas in the columns we present: (a) the mean value and the standard deviation of both the pattern and the alternative solution, (b) the results of the Wilcoxon test "*Z*" that check the statistical significance of differences (we omit the sig. values since for all cases the obtained results have been statistically significant), and (c) the frequency of cases when the pattern "*P*" or the alternative "*A*" have higher metric scores, as well as the frequency of ties "*T*". The cases when one design solution clearly has higher values compared to the other are highlighted with grey cell shading in the table. From Table 5.2 we have excluded the values for ***encapsulation*** (DAM) and ***hier-***

[32] Since the aim of this study is not the evaluation of a specific system, we used the most common values of pattern-related parameters, so that our results to be as generic, and as close to practice as possible.

archies (NOH) attributes, since their scores are equal for both solutions (these metrics are not affected by any pattern-related parameter).

Table 5.2: Effect of Decorator on low-level Quality Attributes

Quality Attribute	**Pattern**		**Alternative**		**Z**	**Solutions**		
	Mean	**STD**	**Mean**	**STD**		**P%**	**A%**	**T%**
Size	13,000	0,09	6,330	1,97	-111,2	99,99	0,01	0,00
Inheritance	0,352	0,16	0,425	0,20	-49,2	38,01	61,92	0,06
Coupling	0,922	0,02	1,307	0,20	-110,8	0,00	100	0,00
Cohesion	0,401	0,16	0,437	0,17	-25,5	40,23	55,18	4,57
Polymorphism	0,615	0,41	0,693	0,52	-32,9	41,66	58,33	0,00
Messaging	5,794	2,49	7,039	2,37	-101,4	8,65	91,27	0,07
Complexity	5,794	2,49	7,039	2,38	-101,3	8,65	91,27	0,07
Composition	0,154	0,02	0,653	0,10	-111,1	0,01	99,99	0,00
Abstraction	1,615	0,26	0,653	0,10	-110,8	100	0,00	0,00

Based on the results of Table 5.2, we observe that for *Inheritance*, *Cohesion* and *Polymorphism* the frequency of occurrences that Decorator has lower metric scores than the alternative solution is close to a 60%-40% distribution. Additionally, concerning *Messaging* and *Complexity* the alternative solution shows 90% higher scores. On the other hand, concerning *Size*, *Coupling*, *Composition* and *Abstraction* the pattern solution has, to a large extent, higher metric scores compared to the alternative solution. A possible interpretation of the higher *Size* (DSC) and *Abstraction* (ANA) values is the increase of the depth of the inheritance tree, and the extra classes placed on the last level of the tree. The result concerning *Composition* (MOA) and *Coupling* (DCC) is intuitive in the sense that in the alternative design the direct composition of Leaf$_i$ to the Decorator was preferred. We note that concerning: (a) some metrics (e.g., *Coupling)* the optimal solution is not the one achieving the highest score, since it is a negative quality indicator; and (b) the same metric can have a different effect on different quality attributes (e.g., DSC is beneficial concerning functionality, but worsens the understandability of the design) (Bansiya and Davis 2002). Finally, the results show that ties are negligible, since they occur rarely (max: approx. 5% for cohesion).

Summing up, the results of the statistical analysis reveal that for *Size* and *Abstraction* the Decorator pattern solution has higher scores than the alternative solution, while for *Coupling, Composition, Messaging* and *Complexity* the opposite applies. Finally, although for *Inheritance*, *Cohesion*, and *Polymorphism* the alternative solution shows more frequently higher scores, the cut-off points split the problem space almost in the middle (60% vs. 40%), suggesting that it is not possible to state if the pattern or the alternative solution is more beneficial, and thus each problem should be individually considered (see Section 5.5.2).

5.5.2 Identification of Cut-off Points

To further investigate the cases where no conclusion can be derived by statistical analysis one needs to work on the model level. By using the equations defined in Section 5.4.5 we subtract the alternative from the pattern function for every quality attribute. In this way, we define a new function (diff) that detects when a solution gets better, with respect to this quality attribute:

$$\mathbf{diff\ (n, p, q, m, k, r)} = \text{pattern}\ (n, p, q, m, k, r) - \text{alternative}\ (n, m, k, r) \mathbf{> 0} \qquad (1)$$

$$\text{diff}\ (n, p, q, m, k, r) < 0$$

The existence of solutions to the aforementioned inequalities (1) suggests that there are multiple cut-off points, where the design pattern solution is getting better or worse than the alternative solution, with respect to a quality attribute[33]. In particular, positive values of diff denote that the pattern version presents higher metric scores, while negative values suggest the opposite. Although in the majority of cases (e.g., cohesion), higher metric scores suggest better levels of the quality attributes, in some cases (e.g., coupling) higher scores imply declined quality. In

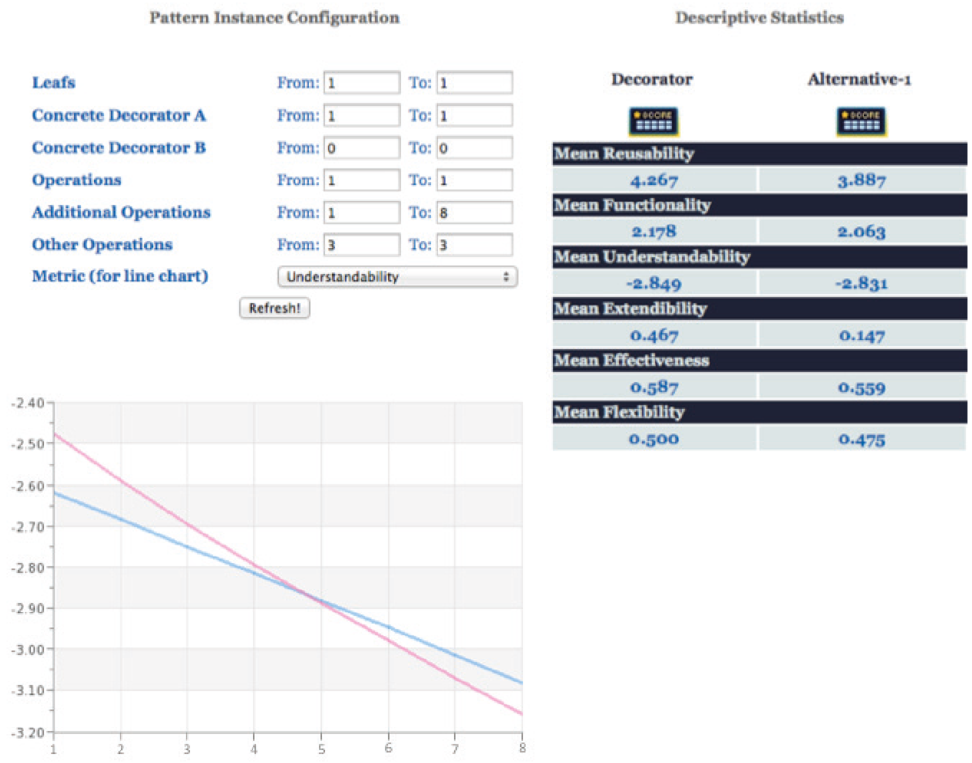
[33] Despite the fact that these solutions cannot be defined as single points, we prefer to use this term to ensure consistency with previous work (Ampatzoglou et al. 2012). In practice the solutions to such equations are cut-off surfaces.

other words, concerning coupling, which has a negative effect on quality, when diff is positive the design alternative is better than the pattern, while when dealing with cohesion, a positive diff implies that the pattern excels. Presenting the mathematical representation of such cut-off points is out of the scope of this manuscript, due to their large number and complexity. Nevertheless, we visualize the existence of these cut-off points by demonstrating a tool created for this purpose.

To assist practitioners in using the proposed method, we have extended the ***DesignPAD tool*** (Ampatzoglou et al. 2012), by adding functionality related to the three newly studied design patterns and by migrating it to the web. Currently, DesignPAD is available as a ***web-service*** through the Percerons platform[34]. The tool requires as input the type of design pattern that the user is interested in (Bridge, Abstract Factory, Visitor, Template Method, State, Strategy, or Decorator), a set of quality metrics or a quality model, and a set of values for the pattern-related parameters (single values or range of values). The tool provides as output descriptive statistics on the metric scores, as well as a visualization of the cut-off points. The results can guide software engineers to make a decision on whether pattern application is beneficial or not.

For example, in Figure 5.3 our method is applied on a Decorator instance with 1 Leaf and 1 Concrete Decorator. In this example the Decorator hierarchy offers 1 polymorphic method and 3 inherited ones, while the Concrete Decorator extends the functionality of the hierarchy by offering 1-8 additional operations. The results of the tool suggest, that the pattern solution gradually becomes more understandable than the alternative, and surpasses it when the solution has 5 additional operations. This finding is according to the intent of the Decorator pattern, which is expected to be useful when adding extra responsibilities to an object (increase of Additional Operations (**r**)). We note that concerning Decorator at this stage the tool is able to simulate instances of only one alternative (the one presented in this study), but in the future we plan to update the tool with further alternatives for all patterns.

[34] http://www.percerons.com



Pattern Instance Configuration

Leafs	From:	1	To:	1
Concrete Decorator A	From:	1	To:	1
Concrete Decorator B	From:	0	To:	0
Operations	From:	1	To:	1
Additional Operations	From:	1	To:	8
Other Operations	From:	3	To:	3
Metric (for line chart)	Understandability			

Refresh!

Descriptive Statistics

	Decorator	Alternative-1
Mean Reusability	4.267	3.887
Mean Functionality	2.178	2.063
Mean Understandability	-2.849	-2.831
Mean Extendibility	0.467	0.147
Mean Effectiveness	0.587	0.559
Mean Flexibility	0.500	0.475

Figure 5.3: Percerons Design Pattern Advisor Output

The most interesting findings on the identification of cut-off points for the Decorator pattern are presented below. We remind that the results correspond to the comparison between the Decorator pattern and the alternative design presented in Section 5.4.1. Functions representing ***abstraction*** (quantified through the ANA metric), ***size*** (DSC), ***composition*** (MOA), and ***coupling*** (DCC) do not present any cut-off points (i.e. the direction of the inequality does not change among different pattern instances) as indicated by the statistical analysis (see Table 5.2).

Concerning ***cohesion*** (CAMC), the obtained results suggest that the larger the number of Leaf$_i$ classes (**n**), the more probable the alternative design solution to become more coherent. Additionally, we observe that as the number of Decorator operation$_i$ methods (**m**) increases the alternative solution becomes more prominent, whereas the opposite applies when adding additionalOperations to ConcreteDecoratorA1$_i$ classes (i.e. increasing (**r**)). This behavior is caused by the addition of the non-coherent methods of a class. For example, in the pattern, operation$_i$ are not coherent with addParts and removeParts. Therefore, as we add such methods, lack of cohesion increases.

During system evolution along two change parameters (adding Leaf_i and operation_i) the use of the pattern leads to less cohesive solutions, whereas when the pattern evolves through the addition of additionalOperations, the cohesion increases.

Next, we present the obtained results regarding the ***Class Interface Size*** (CIS) and ***Complexity*** (NOM). The results on these two metrics are presented together, since their values are equal due to the fact that the pattern does not impose the use of any private or protected methods. For these metrics we can observe that for larger values of (**r**), i.e., adding additonalOperations methods, there are specific combinations of number of classes that the pattern solution offers a larger interface (more methods) per class. Nevertheless, the increase of (**r**) is not the only condition for the pattern solution to exhibit more methods, since the existence of a high number of $\text{ConcreteDecoratorA1}_i$ classes (**p**) is required. This result can be explained by the fact that the addition of extra methods in $\text{ConcreteDecoratorA1}_i$ classes increase the system's average CIS/NOM only in the pattern solution (the changes in $\text{ConcreteDecoratorA2}_i$ are reflected in the alternative as well); thus, the more classes of this role are added, the more the two metrics increase. The existence of public methods is usually considered as a proxy of functionality, and the probability of reusing a specific class in a different system.

Therefore, although small pattern instances (i.e., small number of $\text{ConcreDecoratorA1}_i$ classes (**p**) and additionalOperation methods (**r**)) are offering smaller interfaces than the equivalent alternative designs, along evolution the pattern solution tends to excel in this characteristic.

Concerning ***polymorphism*** (NOP), the only parameter that affects the extent of its use in any of the two designs is the number of classes. Specifically, small numbers of $\text{ConcreteDecoratorA1}_i$ (**p**) and $\text{ConcreteDecoratorA2}_i$ (**q**) lead to limited polymorphism in the alternative solution, and therefore the use of the pattern is preferable. On the other hand, when along evolution more classes are added to the system, the alternative solution takes advantage of polymorphism. However, if the major change is the addition of Leaf_i (**n**), then the pattern becomes more beneficial. This result is expected since polymorphism is present in the Leaf_i classes. Nevertheless, since the use of polymorphism is one of the cornerstones of the object-orientation, designs that make use of it excel in terms of efficiency and extendibility.

Similarly to cohesion, decisions that are based on polymorphism should take into account the most anticipated extension scenarios. Thus, when the number of ConcreDecoratorA1$_i$ classes (**p**) and ConcreDecoratorA2$_i$ classes (**q**) is small and the number of Leaf$_i$ classes (**n**) is large, the pattern solution is beneficial.

Finally, concerning the use of ***inheritance*** (MFA), we can suggest that the addition of operation$_i$ (**m**) and additionalOperation methods (**r**) leads to a more extensive use of inheritance in the pattern solution. On the other hand, the larger the number of otherOperation (**k**) methods, the better the alternative solution becomes. This outcome can be considered as intuitive since when there is no room for the application of polymorphism (all Leaf$_i$ and Decorators have very similar behavior) the use of Decorator, might just be too complex for the designer's needs. Also, the results indicate that some parameters affect more strongly the results. For example, as both (**m**) and (**k**) increase the pattern solution becomes less prominent, which suggests that the effect of (**k**) is stronger, like the aggregate effect of (**r**) and (**k**). Finally, the results when all parameters are increased simultaneously show that the effect caused by the addition of otherOperation (**k**) is stronger than the joint effect of both adding operation$_i$ (**m**) and additionalOperation methods (**r**).

Thus, to understand the effect of Decorator on the use of inheritance one should consider if along evolution the architect expects the addition of operation$_i$ methods that are the same in all Leaf$_i$ and Decorators. As the number of such methods increases, the pattern becomes less beneficial concerning polymorphism.

5.6 Discussion

In this section we discuss the main findings of this study and present implications to researchers and practitioners. In Section 5.6.1 we synthesize our findings to assess six high-level quality attributes, while in Section 5.6.2, we elaborate on the potential value of our method for researchers and practitioners.

5.6.1 Synthesis of Results

To facilitate the discussion on high-level quality attributes, we summarize the main outcomes of Section 5.5, in a synthesized form in

Figure 5.4. In particular, we present six radar charts (one for each high-level quality attribute of QMOOD (Bansiya and Davis 2002)). For each metric that is used to assess a quality attribute we present *the percentage of cases when each design solution is optimal* (PAT: green line, ALT: blue line—by considering the score and

the relation between the metric and the QA), based on the results presented in Table 5.2. We note that from the radar charts we have omitted the metrics that are equal in both solutions (i.e., NOH and DAM). Specifically, the larger the number of metrics that the two lines are close (e.g., CAMC), the larger the gain from using the method, in the sense that the designers can make informed decisions based on the values of the pattern-related parameters.

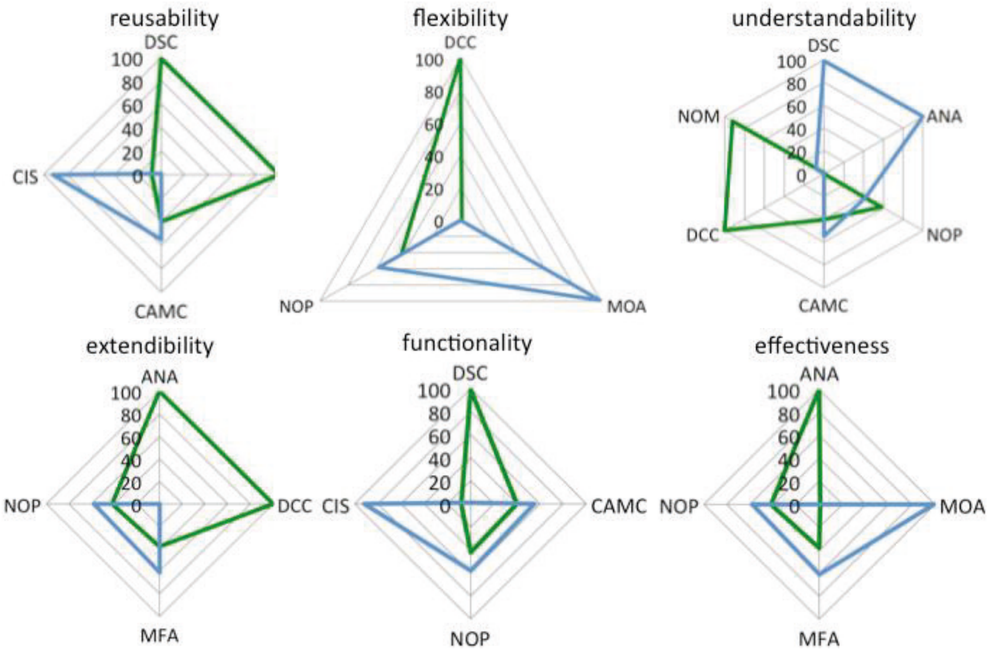


Figure 5.4: Effect of Decorator on Quality Attributes

The aforementioned results suggest that in most of the cases, the application of the pattern enhances the quality attribute of interest. For example, concerning ***Extendibility***, we observe that the design pattern solution improves the values for two out of four metrics. Extendibility is the only high-level quality attribute for which the alternative solution does not excel concerning any factor. This result is in accordance to the literature (Ampatzoglou et al. 2013b), which suggests that Decorator application eases any future maintenance activity. However, there are special cases that some aspects of design quality might be weakened. For example, concerning ***Understandability*** the pattern is always beneficial concerning NOM and DCC. In approximately 40% of the examined cases it is also beneficial concerning CAMC, and in 60% of cases concerning NOP. However, there is no case where the pattern solution is better concerning ANA and DSC. Thus, it becomes clear that since the values of factors influencing understandability are so mixed, we are unable to derive a conclusion on the effect of the pattern using statistics. This result provides a solid explanation on the contradictive results concerning the effect of Decorator on understandability (Ampatzoglou et al. 2011). In particular Ampatzoglou et al. (2011) report that one study has negatively evaluated the effect of Decorator on understandability, whereas two other have reported a positive relation. For such cases further analysis is required.

To assist the process of design solution selection when cut-off points exist, in Table 5.3 we provide a more fine-grained analysis on the factors that influence the effect of patterns on software quality. Specifically, every row of the table presents a metric that is used for the quantification of high-level quality attributes (and presents cut-off points), whereas every column a pattern-related parameter (as presented in Section 5.4.2). Every cell of the table denotes which design is beneficial with respect to the specific metric, whenever one parameter is increasing (i.e., when we extent the system by adding a corresponding class or method). For example, the results of *CAMC* metric indicate that the alternative solution is more beneficial when the number of Leaf$_i$ or the number of operation$_i$ methods increase, while the pattern solution is preferable in the case that the additionalOperation methods increase. We remind that concerning ANA and DSC the pattern solution is always having higher scores than the alternative; regarding MOA and DCC the opposite applies, whereas for DAM and NOH they are always equal (see Section 5.5.1).

Table 5.3: Effect of Decorator Parameters

Metric	**Modification Parameters**					
	(n)	**(p)**	**(q)**	**(m)**	**(k)**	**(r)**
CAMC	ALT			ALT		PAT
CIS		PAT				PAT
NOM		PAT				PAT
NOP	PAT	ALT	ALT			
MFA				PAT	ALT	PAT
Total	2	3	1	2	1	4

Based on Table 5.3 and the radar charts of Figure 5.4 we can guide practitioners in making pattern-related decisions, based on their preference on different quality attributes as follows:

Reusability. We can observe that 2/4 metrics that influence reusability (DSC and DCC) are always favored by the use of the pattern. Concerning the other two (CIS and CAMC), we can observe that in majority the alternative design is more beneficial. However, in the special case that along evolution, the practitioner expects an increase in the number of concrete decorators (**p**), which offer increased number of class-specific operations (**r**), then the use of the pattern seems like a better choice.

Flexibility. One metric (DCC) supports the use of the pattern, another (MOA) supports the alternative, and one (NOH) is neutral. The fourth metric that presents cut-off points (NOP), shows a balanced behavior. The use of the pattern can be suggested when more types of components are expected to be added inside the decorator container (**n**), or more concrete decorators that offer class-specific operations (**p**). Nevertheless, according to Di Penta et al. (2008) adding classes to an existing Decorator instance is not the most frequently applied modification operation. This observation can partially explain the negative effect of Decorator on adaptability, reported in the literature (Ampatzoglou et al. 2013b).

Understandability. Similarly to reusability, the existence of cut-off points is important, since 2/6 relevant metrics (DCC and NOM) are always positively affected by the use of the pattern and two metrics (DSC and ANA) are always favored by the alternative. For the rest (CAMC and NOP), we observe that adding concrete decorators that offer class-specific operations (**p**) makes the pattern more beneficial in terms of understandability, whereas adding concrete decorators that do not offer class-specific operations (**q**) or operation$_i$ methods (**m**), favor the application of the alternative solution.

Functionality. Concerning this quality attribute only one metric (DSC) is always positively affected by the pattern, and three others (CAMC, NOP, and CIS) exhibit cut-off points. The rules that apply for functionality are the same as for understandability (high number of ConcreDecoratorA1$_i$ classes (**p**): benefit from pattern, high number of ConcreDecoratorA2$_i$ classes (**q**) or operation$_i$ methods (**m**): benefit from alternative).

Effectiveness. This quality attribute is related to two metrics that present cut-off points (MFA and NOP). These metrics, in most of the cases, benefit from the alternative design. However, they are influenced by completely different parameters (NOP is influenced by class-related parameters, whereas MFA by method-related parameters), and therefore, they cannot be discussed uniformly and every evolution scenario should be treated individually. For the other two metrics that influence effectiveness one favors pattern (ANA) application and other the alternative (MOA).

Extendibility. This is the only quality attribute that the alternative solution does not present higher scores for any of the metrics that influence it. Therefore, we can assume that for the majority of the cases the design pattern solution can be more easily extended. The two metrics presenting cut-off points (MFA and NOP) are exactly the same as in the case of effectiveness and therefore the same observations apply.

5.6.2 Implications to Researchers/Practitioners

Based on the aforementioned discussion on the effect of the Decorator pattern on quality attributes, we can highlight that design quality is diminishing by the addition of concrete decorators that do not offer class-specific operations (**q**) or methods that are common in all decorators (**k**) and in such cases alternative designs should be preferred. A possible explanation is that these types of change do not conform to the rationale of the pattern. For example, if the majority of methods that exist in the hierarchy are the same, then its benefit is limited to a small number of polymorphic methods. The results of the study lead us to some useful implications for researchers and practitioners, as follows:

- Researchers can use the proposed method (subjected to some modifications) for studying similar issues in the design phase, e.g. formulating the effect of refactorings on software quality.
- Researchers can generalize the method so as to be able to compare equivalent design solutions, across software evolution, regardless of pattern participation.
- Researchers can use the proposed analytical method for investigating the effect of patterns on source code metrics.
- Practitioners can use the derived formulas for making design decisions during both *Greenfield* and *Brownfield* development. In the first case (during design) the designer can consider factors, like the number of the pattern-participating classes of an instance to decide prior to the application of a pattern whether this would be beneficial. In the case of *Brownfield* development, the same approach

can be used during the maintenance phase, for scheduling a refactoring of a pattern-based solution to an alternative one, or vice versa. In both cases the obtained benefit is the capability to evaluate pattern-related design decisions before they are implemented, contributing to reduced development or maintenance costs.

5.7 Threats to Validity

In this section we discuss threats to validity. Concerning *construct* validity, the mapping between quality attributes and metrics, as provided by QMOOD, is acknowledged as a threat. However, QMOOD has been rigorously validated during its introduction (Bansiya and Davis 2002). Nevertheless, we note that the riskiest part of the model (i.e., assignments of weights to low-level metrics) has been omitted. Additionally, the conducted experiments do not necessarily capture the construct of design evolution accurately, since it is possible that design may evolve in certain directions, but our sample scenarios count as if they are all equally probable to happen. Thus, it is possible some of the generated data points to represent infeasible evolutions, but contribute equally to the results.

In terms of *external* validity, the use of the QMOOD suite certainly poses some threats, since the use of a different model might produce different results. Similarly, the generalizability of our results is influenced by the use of specific design alternatives, expecting that alternatives with poor design could result to even better scores for the pattern solution. However, we note that the applicability of the method depends neither on the use of the selected model nor the selected alternative. The method can be used with any metric suite that takes into account some pattern parameters (e.g., (Chidamber and Kamerer 1994)), as well as with any alternative solution that is equivalent to a GoF design pattern; the selection of the design solutions depends on the judgment of the software engineer who applies the method. Thus, we do not imply that the selected alternative is the best Decorator alternative; after all there is no objective way to compare all available solutions.

The study has limited *reliability* threats, since all research questions were answered by mathematical operations, which involve no researcher bias. Although, the selection of the pattern related parameter ranges is subjective, it is based on empirical results obtained from OSS development. Finally, *internal* validity may be influenced by the pattern related parameters selection, in the sense that omitted parameters can be considered as confounding factors. However, in this study we selected to explore the most frequently changing parameters, according to Ng et al. (2007).

5.8 Conclusions

This chapter presented a study which aimed at developing a method that can provide guidance to designers while making pattern-related decisions, driven by qualities. The results of applying the method on decorator highlighted that in most cases pattern application is beneficial for the design-time qualities; however, there are specific cases when alternative solutions should be considered. In particular, we provided evidence that when the decorator pattern is applied in the right context, i.e., many concrete decorators, with high variability of offered functionalities (methods), it positively affects quality. On the other hand, in cases that the pattern is extended by concrete decorators, which inherit most of their offered functionalities, some quality attributes diminish. Based on the above we can claim that the provided method can be useful to practitioners, and at the same time it opens some interesting research directions.

This chapter addressed the second limitation of the problem statement of this thesis, i.e. the lack of systematic support for identifying incorrectly instantiated patterns, as well as the lack of guidance on how to refactor the design for the purpose of repaying design TD. In the following chapters we will shed light on the third limitation which is related to another TD type, i.e., the Documentation TD.