A Composite Design-Pattern Identification Technique

Marjan Heričko and Simon Beloglavec
University of Maribor, Institute of Informatics,
Smetanova ulica 17, 2000 Maribor, Slovenia
E-mail: marjan.hericko@uni-mb.si, simon.beloglavec@uni-mb.si

Keywords: Design patterns, design metrics, design patterns identification, composite design patterns

Received: June 17, 2005

This paper introduces a new technique for identifying composite design patterns from existing pattern-based designs. We propose two pattern metrics: pattern coverage and overlapping that can help detect a composite pattern. The effective composite patterns reflect quality properties that are considered desirable in the solution for a given problem domain and selected programming paradigm. To identify appropriate candidates, we propose an assessment with a set of design metrics in addition to pattern metrics. The calibration of value intervals for metric scores is proposed with the intention of offering the designer the possibility of adjusting the technique for each individual type of software. In this paper, we present the steps required for detecting and identifying the suitable composite pattern candidates through pattern and design metric assessment.

Povzetek: Prispevek predstavlja nov pristop z novimi metrikami vzorcev k identifikaciji sestavljenih načrtovalskih vzorcev v obstoječih načrtih informacijskih sistemov.

1 Introduction

The typical software design rarely includes an independent pattern; increasingly, applied patterns are interconnected. A design pattern (henceforth "pattern") can be applied to various structural forms. A set of applied patterns, in selected forms, can promote in the existing designs desired quality characteristics. What qualifies as an appropriate design quality depends on the type of software that has been developed (e.g. local component, distributed component, programming library, etc.). Therefore, in some cases a set of patterns proves to be an efficient solution while in other cases it results in unwanted design complexity. The designer's goal in a pattern-based design is the application of an effective pattern combination. The proven solutions of pattern applications can be identified from existing designs.

We propose a composite pattern identification technique that consists of three main steps. The first step towards the identification of suitable composite patterns is the construction of the pattern coverage matrix for the selected design. The matrix holds information over the selected pattern instantiation form. The instantiated form is one of the allowable forms of a pattern that includes all allowed structural and behavioural variations for the selected pattern. The information over the instantiated pattern form captured in the pattern coverage matrix contains a detailed description over the selected structural and behavioural variations that are applied in a design. The constructed matrix is then assessed with the pattern coverage metric that is defined in this paper. The goal of the assessment is the identification of design fragments that are covered with patterns. During the second step, we construct a pattern overlapping matrix based on the pattern coverage matrix. In this paper, we define a pattern overlapping metric that is intended for detecting various levels of overlapping. This step extracts the set of composite patterns candidates that is assessed with design metrics in the final step. The final assessment uses a set of design metrics that exposes flaws in the design when considering quality attributes valid for a given solution domain and the selected programming paradigm. The result of the final stage is a small subset of new composite patterns or an individual composite pattern. A possible outcome is also an empty acceptable set from the set of extracted pattern candidates. Identified composite patterns act in future applications equal as atomic patterns.

The application of the technique is presented in two design cases where composite patterns are identified. The paper demonstrates how the proposed technique applied on the first simplified design detects the well-known composite pattern (MVC- Model-View-Controller pattern) from an existing design. The second example demonstrates the technique's application through a complex design where the calibration of value intervals for metric scores is presented in detail and a new composite pattern is extracted.

The rest of the paper is structured as follows: In Section 2, relevant background and related works are discussed. Section 3 contains the steps of the technique and defines the proposed pattern metrics for coverage and overlapping. Section 4 demonstrates the application of the technique through the identification of the MVC pattern from a design. An approach to the calibration of value intervals for design metric scores is discussed in Section 5. Section 6 gives some conclusions and ends with a discussion of the research findings.

2 Background and Related Works

To avoid ambiguity when discussing patterns, it is important that we define the term composite pattern and also define the types of patterns that are suitable for the application of the technique. The composite pattern [19] refers to a composition of patterns that have a common solution space and is not to be mistaken for the design pattern from a fundamental catalogue [1]. Patterns can be classified in many ways: lifecycle stage (requirement, analysis and design patterns) and level of abstraction (idioms, design and architectural patterns). This research focuses on design patterns and makes a clear distinction between an atomic and a composite pattern. Atomic patterns are considered to be the fundamental patterns, which build a pattern language and cannot be broken down into a set of sub-patterns. Composite patterns are a product of pattern integration that go beyond a simple composition that groups patterns without any synergy [16]. The existing research defines composite patterns in various ways. Some researchers consider a composite pattern to be a set of patterns from various architectural levels (analysis, design, implementation) [18], others focus on the dependencies between applied parts of the patterns in the design [16], [17] or on compositions that are discussed in a pattern catalogue level without considering the target design [20]. The fundamental pattern catalogue [6] also defines a set of relations that can be treated as connections in a composition. The presented technique focuses on the patterns applied into a design and considers the pattern overlapping that can be present in specific design parts. Overlapping occurs when an individual design part has a role in two different patterns. Composite patterns that are identifiable with the presented techniques all have constituents as overlapped patterns. The identification technique starts the analysis from the instantiated pattern variant in a design. The specific treatment of pattern overlapping distinguishes the presented approach from other existing attempts at composite pattern identification.

Some of the early attempts at identifying patterns from an existing solution were built exclusively from the structural information that was constructed from a source code. The fundamental presumption in such research has been that pattern extraction is possible without additional information. Many authors ([12], [14], [16] and [22]) use object-oriented software metrics for the purpose of identifying structural GoF patterns [6]. In the case of other pattern types, false positives can occur ([12] and [16]). False positives must be detected and inspected by the user alone. Single class metrics are used to reduce the search space in a structure. In previous research, metrics such as NOA (Number of Attributes) and NOO (Number of Operations) have appeared in various configurations. Metric scores are used for the detection of candidate classes for structural patterns. The similar usage of metrics, for detecting the structure of fundamental patterns, has been tracked by various authors [11], [16].

Patterns can be detected with the help of basic metrics on the class structure. A question has arisen in the past: does the application of patterns have an influence on software quality metric scores? In many cases, patterns promote weak coupling between classes and a greater abstraction if the impact is observed on the level of an individual pattern [7], [21], [9]. A comparison has to be carefully made while also considering various influences (other patterns, external non-pattern classes). The process of detecting composite patterns can return different results, and should be assessed on adjusted score intervals, as shown later in the paper. Design metrics, if applied properly, have proven effective as indicators of flaws and the inappropriate use of patterns in existing designs [23].

The domain and language-independent discovery of patterns is possible with the use of formal specifications, which serve as an independent meta-layer between a specific design and conceptual artefacts. A formal specification language enables the formal definition of the patterns themselves and their application [1][5]. The independence from a design paradigm is not pursued in all research [4]. While in most cases, the analysis of a source code is the leading source of data, some researchers also decided to include the data over behaviour during system run-time [7]. A demanding construction procedure with such specifications prevents researchers from utilizing other approaches. The presented technique does not require such specifications.

3 Proposed Technique

New editions of pattern catalogues have motivated the quest for discovering new design patterns. The expression discovery process can be ambiguous. Some research uses the expression discovery, when actually a recovery of well-known patterns is done. The identification of patterns using the proposed technique results in new composite patterns. In the presented case, we analysed existing solutions where we presumed that proven composite patterns are present. The technique is meant to be applied in cases that have already proven to be successful in the real world. We use the term identification instead of discovery, in order to stress the fact that in presented cases, composite patterns are already present and only need to be identified. Applying the technique enables the designer to select candidates from a design and identify the appropriate ones, considering the positive properties for the selected programming paradigm. The pattern-based design preserves the information on applied patterns (instantiated pattern variants and their locations in a design). The goal of the identification process in all cases is to detect the patterns that can be atomic or composite. Atomic patterns are not a result of composing existing patterns. Early research dealt with the discovery of atomic patterns, which are included in existing catalogues. Finding a new extracted pattern that can be used in future designs, like any other pattern, justifies the invested effort. The application of a composite pattern increases the pattern's usage and protects a designer from the inappropriate application of several patterns. The set of patterns can be applied in a design with many

variations, while the composite pattern consists of a proven solution for their application.

A single pattern can appear in different designs in many variations. Pattern catalogues suggest basic forms of a pattern while possible variations are rarely discussed in detail. Some parts of a pattern can be omitted without compromising the mission of a pattern. For example, the pattern Lightweight [1] can in some cases includes the classes that represent the unshared concrete flyweight, while in other cases these classes are omitted. In some cases, the same building element appears in different shapes. For example, the Flyweight pattern itself can be described with an abstract class or with an interface. It is to be expected that the same patterns will have a different cardinality and types of elements. This fact does not directly interfere with the presented technique. This fact should be considered during the construction of the input data for the technique. The use of a standardized template, with fixed elements for each individual pattern, is not adequate in our approach.

The variety of formats tilts many reengineering and assessment projects away from specifying patterns in their design [3], [8], [10], [15]. The information in the applied patterns is a valuable base for further analysis. The purpose of the presented method is not to identify pattern candidates through the structural information that is constructed from the program's source code. A base consists of information on a pattern's variants that are applied in a design. If existing designs preserve information over the applied patterns, we can extract the necessary data to apply the technique. In order to automate the whole procedure, a mapping facility must be constructed that translates the pattern information into the form required by the proposed technique. We avoided building a meta-level specification (formal or informal) in this research. Existing designs, known to authors, use a variety of semi-formal and formal notations for describing applied patterns. The motivation that drove this research was establishing the minimal denominator of the pattern information, where construction is feasible in all known cases.

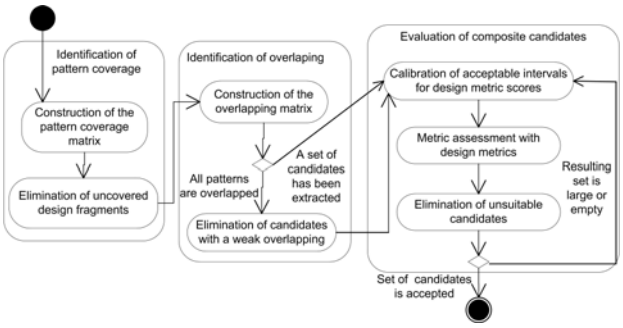

Figure 1: Activities for a composite pattern identification

In order to perform the technique presented in Figure 1, the input data must be prepared in a prescribed manner. For all the patterns used in an observed design, the distinct variants of the pattern application should be identified with all the corresponding parts. We presume that the existing specifications of an analysed design will allow us to identify the pattern parts in the design at the detailed level of methods and attributes. The pattern coverage matrix needs to be constructed in order to perform the remaining steps. The matrix values are calculated as pattern metric scores. The pattern coverage metric is defined in the following chapter. The values in the matrix enable the elimination of uncovered design parts from further analyses. They also constitute the base for detecting the overlapping of applied patterns, as calculated and assessed in the overlapping matrix. Only the parts of the design that are actually covered with patterns should be considered. Other parts are not important in the further identification process. The matrix data on pattern coverage serves for the detection of pattern overlapping. The reasoning behind treating overlapping as a key data in the discovery process is explained in Section 4. In some cases, analyses of the pattern overlapping matrix produces only one composite pattern candidate that includes all patterns, which appear in the design. To avoid the extreme case of accepting a whole design as a pattern, the strength of overlapping should also be inspected. Later in the paper, we define the strength levels for overlapping. Patterns with weak overlapping can be eliminated from the candidate pool. If all patterns are connected with the same strength of overlapping, this combination becomes the only composite pattern candidate. The type of software that is being developed dictates the attributes, which can be expressed through design metric scores. When multiple candidates are present in a set of detected composite patterns, the design metric assessment eliminates the unsuitable candidates. The assessment is also reasonable in cases when there is only one candidate for a composite pattern. The purpose of the assessment is to examine the candidates' suitability with regard to the quality characteristics implied by a solution domain. The technique does not behave as a decision function that result in one candidate only. The number of final candidates depends on the calibration of allowed value intervals for metric scores. The designer's decision is to accept all the positive candidates or only the most appropriate ones considering the metric scores.

Designers try to avoid the realization of the following statement: "Patterns usually lead to an increased number of software artefacts, which normally increases the static complexity of a software system considerably" [23]. A high level of overlap in a pattern prevents the undesired increase of artefacts. Upholding this level forces the designer, with each new pattern application, to integrate a new pattern well into the design.

There is no standardized definition for the "glue" between patterns in a composition. If the connecting glue is presented by the interaction-dependency between the pattern parts of various patterns there are as many candidates to be considered as the composites [16]. If the analysis encompasses the abstraction level of an interface (all public patterns are taken into consideration) or an implementation (all detailed structures are considered), an excessive amount of interaction is to be expected. Observing patterns as a whole in a design, it appears that

all the patterns are connected through some interactions. An alternative presents the relationships that are defined by the pattern catalogue. Using these relationships between patterns, like glue in a composite, significantly reduces the possible combinations. However, no standardized set of pattern relations is defined when considering multiple catalogues. This reduces the space for pattern detection on an individual pattern language with the presumption that there is an appropriate set of relationships available. There is also another downside to this approach – instantiated pattern variants are not considered. We followed the idea in the statement: "Integrated patterns should show synergy that makes the composition more than just the sum of its parts" [16]. Our interpretation of a pattern synergy concept is as follows: The individual pattern parts in a composite should provide more pattern functionality than they provide when applied separately. The guideline for good synergy between patterns, in a composition, can be found in the level of pattern overlapping. The patterns in a composite can overlap. An individual part of such a design has various roles in used patterns. Overlapping can be observed on all the building parts of a pattern that are suggested in a pattern definition. A high level of overlapping indicates strong integration between individual patterns. Henceforth, we will define composite patterns as a set of patterns that are connected with the overlapping parts. When overlapping between patterns is detected, the candidates for composites can be extracted.

The data needed for pattern coverage and pattern overlapping presentation requires that patterns applied in a design be conceived as sets of the connected building elements, which include classes, interfaces, methods and attributes. The methods and attributes, which are prescribed by a pattern, present the building parts for pattern classes and pattern interfaces. Classes and interfaces are referred to as the main elements of a pattern or a design, while the methods and attributes of a class or interface are referred to as sub-elements of a pattern or a design. The pattern coverage matrix precisely defines the form of instantiated patterns in individual design fragments. The matrix can be presented on a whole pattern, an element or a sub-element level of detail. For reasons of clarity, we will present only a small fragment of the sample design on a detailed level.

Let $p^s = <e^s_1,...,e^s_i>$ be a pattern p^s where e^s_x is an element of the pattern p^s. For each e^s_x there are an array of sub-elements $e^s_x = <s_1,...,s_j>$ where e^s_x is a sub-element of the pattern p^s. The design can be presented in a similar way. Let $d=<e^d_1,...e^d_m>$ be a design or a design fragment. For each e^d_x there are an array of design sub-elements as in the pattern $e^d_x=<s^d_1,...,s^d_n>$. A main element of a design (class or interface) can be covered with the multiple pattern elements that belong to various patterns. In the overlapping matrix, the columns represent pattern parts, while the rows represent design parts. The matrix can be presented through various detail levels, which reveal the pattern coverage on a level that is appropriate to perform analyses. On a sub-elemental level, the matrix values can only be presented with the values of 0 or 1. The value 1 means that a sub-element of the pattern is instantiated in the sub-element that is presented in a matrix row. On the main elemental level, the idea is to determine how many pattern sub-elements (attributes and methods) cover the main element of a design. The value is the sum of the coverage. On the whole pattern level, the values as expected represent the sum of all main element coverage. The previously described coverage values are defined by the following formulas:

$$(1)\ \mathrm{cov}_{sub-sub}(s^d_x, s_y) = \begin{Bmatrix} 1 & s_y \rightarrow s^d_x; s^d_x \in d \wedge s_y \in p \\ 0 & otherwise \end{Bmatrix}$$

$$(2)\ \mathrm{cov}_{main-sub}(e^d_x, s_y) = 1 + \sum_i \mathrm{cov}_{sub-sub}(s^d_i, s_y)$$

$$(3)\ \mathrm{cov}_{main-main}(e^d_x, e_y) = \sum_i \mathrm{cov}_{main-sub}(e^d_x, s_i)$$

$$(4)\ \mathrm{cov}_{main-pattern}(e^d_x, e_y) = \sum_i \mathrm{cov}_{main-main}(e^d_x, e_i)$$

Formula 1 is used to determine coverage between the sub-elements of a particular pattern and the sub-elements of a design. The value 1 in formula (2) is added because a class or interface should also be counted as an element. For representing the matrix in all coverage details, the following formulas are also necessary:

$$(5)\ \mathrm{cov}_{sub-main}(s^d_x, e_y) = 1 + \sum_i \mathrm{cov}_{sub-sub}(s^d_x, s_i)$$

$$(6)\ \mathrm{cov}_{sub-pattern}(s^d_x, p) = \sum_i \mathrm{cov}_{sub-main}(s^d_x, e_i)$$

The coverage on the whole design is not important because it results in the number of all pattern parts in a pattern. Thus, it is meaningless, since we are interested in those parts of a design that are strongly related to applied patterns. Pattern coverage (cov) is the first of the two pattern-based metrics we proposed in this paper. To demonstrate the use of the defined coverage metric, we will use a sample design, presented in Figure 2. As we can see, the well-known MVC [13] composite pattern has been applied. The MVC pattern integrates three atomic patterns: Observer, Strategy and Composite [1].

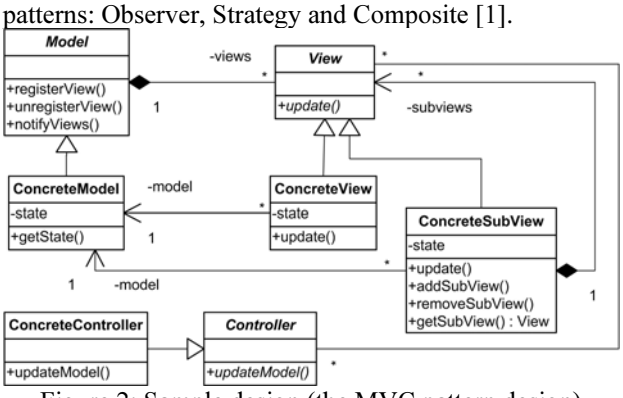

Figure 2: Sample design (the MVC pattern design)

Design / Pattern (*cov*)	Composite	Observer	Strategy	*Context*	*Strategy*	*Concrete Strategy*
Model	**2**	**5**	**0**	*0*	*0*	*0*
ConcreteModel	**0**	**3**	**0**	*0*	*0*	*0*
View	**2**	**2**	**2**	*1*	*0*	*0*
ConcreteView	**2**	**4**	**0**	*0*	*0*	*0*
state	*0*	*1*	*0*	*0*	*0*	*0*
model	*0*	*1*	*0*	*0*	*0*	*0*
update	*1*	*1*	*0*	*0*	*0*	*0*
ConcreteCompositeView	**6**	**4**	**0**	*0*	*0*	*0*
Controller	**0**	**0**	**1**	*0*	*1*	*0*
ConcreteController	**0**	**0**	**1**	*0*	*0*	*1*

Table 1: Pattern coverage for sample design

Table 1 contains pattern coverage values with various level of details for a sample design (Figure 2). With the previously defined formulas (1-6) we calculated table-cell values only. The main design element ConcreteView is presented on a sub-elemental level of details. The pattern Strategy is presented on the main-element level. We propose that the level of detail be adjusted by the designer, as regards the desired clarity level of the presentation. The pattern coverage matrix that is presented in an appropriate level of detail proves useful when presenting how parts of a pattern are instantiated in a particular design.

From the main-element level of details for the pattern Strategy, we can notice that the design class View represents the context in the Strategy pattern, for which different strategies can be available. In the sample design, only one concrete strategy is present and is instantiated in the design class "ConcreteController". The basic behavior of the strategy is defined in the pattern with the class Strategy that is instantiated in the design class "Controller". The inspection of the sub-elemental level of detail for the design class ConcreteView shows that the attributes "state" and "model" have a role in the pattern Observer, while the update() method appears to have a role in both the Composite and Observer patterns.

As presented, some design elements are covered with multiple patterns. We use the term pattern overlapping in cases where an individual design part is covered with multiple patterns. Pattern overlapping can be observed, in a similar way as pattern coverage, in various levels of detail. Pattern overlapping is meaningful when observed in different patterns. Let s_x, s_y be a sub-elements and e_x, e_y a main-elements of distinct pattern applications p^x, p^y for a same pattern:

$$\forall x, y; s_x, e_x \in p^x \wedge s_y, e_y \in p^y \wedge x \neq y .$$

If in a design there are two applications of the same pattern, these patterns are considered as different and an overlapping value can be calculated. We applied the following formulas:

$$(7)\quad ovl_{sub-sub}(s_x, s_y) = \begin{cases} 1 & s_x \to s^d \wedge s_y \to s^d; \exists s^d \in d \\ 0 & otherwise \end{cases}$$

$$(8)\quad ovl_{sub-main}(s_x, e_y) = 1 + \sum_i ovl_{sub-sub}(s_x, s_i)$$

$$(9)\quad ovl_{main-main}(e_x, e_y) = \sum_i ovl_{sub-main}(s_i, e_y)$$

$$(10)\quad ovl_{main-pattern}(e_x, p^y) = \sum_i ovl_{main-main}(e_x, e_y)$$

$$(11)\quad ovl_{pattern-pattern}(p^x, p^y) = \sum_i ovl_{main-pattern}(e_x, p^y)$$

Formula (7) defines overlapping on its basic sub-elemental level. In formula (9) we provided a joint formula for the overlapping of the two main pattern elements. Overlapping is also assessed on a whole pattern level in formula (11). The remaining formulas (8) and (10) enable the calculation of a presentation on various detail levels.

Pattern (*ovl*)	Composite	Observer	Strategy
Composite	**-**	**10**	**1**
Observer	**10**	**-**	**1**
ObserverPart	*2*	*-*	*0*
ConcreteObservedPart	*0*	*-*	*0*
Observer	*2*	*-*	*1*
ConcreteObserver	*6*	*-*	*0*
Strategy	**1**	**1**	**-**

Table 2: Pattern overlapping matrix for the sample design

Table 2 lists scores for the overlapping metric. The pattern Observer is shown on a main-element level of detail. To express how strong the overlapping is between two patterns we define a pattern metric, the overlapping factor. Let n_{px} and n_{py} be the number of all the pattern parts (main and sub-elements) for the patterns p^x and p^y. The overlapping factor fovl between these patterns can be expressed as:

$$(12)\quad fovl_{pattern-pattern}(p^x, p^y) = \frac{ovl_{pattern-pattern}(p^x, p^y)}{n_{px} + n_{py}}$$

Pattern (*fovl*)	Composition(15)	Observer(14)	Strategy(4)
Composition	-	0,34	0,05
Observer	-	-	0,06
Strategy	-	-	-

Table 3: Pattern overlapping factors

Table 3 shows values for the factor of overlapping that is calculated on the base of results from the Table 2. The scores show that if the pattern p^x overlaps with the pattern p^y it is also true that p^y overlaps with p^x. For this reason, we omit a redundant calculation of these elements if the table is observed as a matrix. The numbers of pattern parts are stated in brackets near the pattern name. The results show that in the MVC pattern all elements are connected through overlapping. The overlapped patterns are the appropriate candidates for new composites.

4 The Overlapping Detection

In the previously presented sample design, the MVC pattern has been detected. The calculated values for the overlapping factor show different strengths between used patterns. These strength levels can serve for the extraction of smaller pattern candidates that show high integration, if overlapping factor is considered. The following example is a design with five applied patterns. The intention is to demonstrate a possible reduction of a pattern candidate's size in the situation where all pattern parts appear to build a single composite pattern. From the patterns applied in a design, the designer should identify the suitable composite pattern candidate that appears to

have the strongest overlapping between involved patterns.

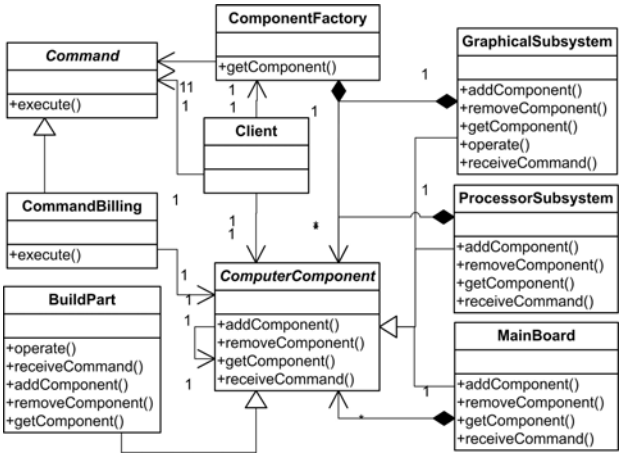

Figure 3: Sample design (the MVC pattern design)

Figure 3 shows a design for the bill of material component (BoF). The following patterns are applied: Decorator, Command, Composite, Visitor and Flyweight. The Composite pattern enables the building of a composite BoF. There are the three possible compositions that can appear in the BoF: "GraphicalSubsystem", "ProcessorSubsystem" and "MainBoard". A final leaf component is represented by the instances of the class "BuildPart". The client façade is presented by the class "Client". The Façade pattern is not explicitly exposed explicitly in the further analysis. The Flyweight pattern introduces a pool of instances for the building parts. This prevents the redundancy of objects that construct a large BoF. The Decorator pattern is introduced to later enable a dynamic adding of functionality to the class "ComputerComponent". The Visitor, in combination with the Command, enables the execution of individual calculations of the individual building parts for the BoF. To extract the most suitable composite pattern it is recommended to isolate parts with a high level of overlapping.

Overlapping / Patterns	Dekorator	Command	Composite	Visitor	Flyweight
CommandBilling	0	2	0	2	0
Command	0	2	0	2	0
Client	1	5	2	0	4
ComponentFactory	0	0	0	3	3
ComputerComponent	3	0	4	1	1
BuildPart	0	1	2	3	2
MainBoard	1	1	5	5	4
ProcessorSubsystem	1	1	5	5	4
GraphicSubsystem	1	1	5	5	2

Table 4: Pattern overlapping matrix

Table 4 shows the pattern coverage in the given component design. A brief analysis of the calculated values indicates a strong overlapping in some cases. To distinguish between different overlapping levels, we propose following value intervals for pattern overlapping factors that can present a base for the classification. We have defined three levels of overlapping: weak {0<*fovl*<0,3}, medium {0,3<*fovl*<0,5} and strong {*x*>0,5}. The intervals were defined based on our experiences and an analysis of various designs. The scores for the detected MVC pattern in the previous example indicate a weak overlap between the pattern Strategy and the other two patterns. A medium overlap exists between the patterns Composition and Observer. Reduction should be considered in cases where the candidate pattern appears to be over-specialized. The trash point should be determined by the designers, based on their experience.

Pattern (*fovl*)	Dekorator	Command	Composition	Visitor	Flyweight
Dekorator	-	0,16	0,21	0,26	0,23
Command	-	-	0,24	0,38	0,43
Composition	-	-	-	0,52	0,79
Visitor	-	-	-	-	0,89
Flyweight	-	-	-	-	-

Table 5: Pattern overlapping factors

Table 5 shows pattern overlapping factors for the BoF components. In some designs, such a table can become large and unclear. To achieve a clearer overview we propose a graphical representation of the overlapping levels.

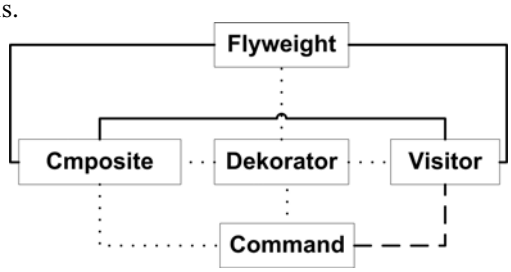

Figure 4: Graphical representation of overlapping levels

The lines that connect the patterns show the strength of the pattern overlap. In Figure 4, weak overlapping is indicated with a dotted line, medium with a dashed one, and strong overlap with a solid line. In the presented case, the Decorator pattern can be omitted from the composite pattern candidate if weak overlapping is not considered. To confirm the suitability of the composite pattern candidate, a design metric assessment should be performed.

In some cases, multiple existing designs have to be reviewed and analysed and the designer has to select suitable composite pattern candidates. If various levels of strength in overlapping are detected, then only the patterns connected with a medium or strong overlap should be considered in the further analysis.

5 Assessment of Candidates

According to the proposed technique, composite pattern candidates should be validated in the final stage. Validation is performed in the form of an assessment with the selected design metrics. The acceptance criteria should be defined based on the design metric scores that are specific for the solution space and the targeted type of software. The metric assessment eliminates unsuitable candidates in the final stage of the composite pattern identification procedure. The interval for the individual metric has to be calibrated to meet the expected property values for the given solution space and design paradigm. The sets of metrics are specific for the individual programming paradigm. Selected metrics in a set vary regarding the type of software that is developed.

The metric assessments used in an appropriate design stage help detect weaknesses in a design. Their application in the re-engineering phase helps to analyse the suitability of the design fragments. Only metrics that are influenced by the pattern application are stressed. Patterns, if assessed individually, promote a weak coupling and higher abstraction levels, which reflects on metric scores. Expected scores should reflect the desired qualities for the type of software (for a given problem domain and/or solution space). We propose a calibration of the targeted acceptance intervals for the each particular case. Defined intervals should reflect the properties that are expected to be met. For example: patterns that help build individual components should allow inherent coupling, and promote re-usability of the whole structure instead of re-usability on an individual class level. To prevent the influence of non-pattern elements, the design metric assessment is performed on isolated design fragments. Those that are influenced by a pattern application in the design phase of software development.

6 Conclusions

This paper presented the technique for identifying composite patterns in existing pattern-designs. The identification process encompasses various metric assessments. We have introduced two pattern-based metrics that enabled us to assess design fragments. While other existing researchers propose pattern identification through source code metrics, the presented technique performs assessments on the pattern level. With a sample design, we have demonstrated that the technique is also able to identify well-known composite patterns such as MVC. The identification of composite patterns, i based on pattern metrics, can result in multiple pattern candidates. To confirm if the given candidates are suitable, an additional assessment with design metrics was proposed. The goal of this assessment was to identify the most suitable candidate. A designer specifies acceptable intervals for selected metric scores that reflect the properties of a design fragment. The final result of performing the steps of the technique is composite candidates with metric scores within acceptable intervals. We have demonstrated a sample calibration of intervals for metric scores with the sample design of a component.

Through the application of the presented technique, new composite patterns can be identified in existing designs. Identified patterns can enhance the existing fundamental catalogues and provide good practice for how to apply a group of atomic patterns in similar solution spaces. This technique distinguishes itself from existing approaches of pattern identification through the use of combined assessment with pattern and design metrics. The technique can also be modified for the identification of composite anti-patterns. An additional repository of anti-patterns could prove useful in forward engineering, when the composition of patterns is required.

References

[1] J. Bazan, J. F. Peters, A. Skowron, N. Hung Son, M. Szczuka, Rough set approach to pattern extraction from classifiers, Electronic Notes in Theoretical Computer Science, Volume 82, Issue 4, March, 2003, p. 1-10.

[2] S. Chidamber, C. Kemerer, A metric suite for object-oriented design, IEEE Transactions on Software Engineering 20(6), 1994, p. 476-493.

[3] J. Dong, Adding pattern related information in structural and behavioral diagrams, Information and Software Technology, Volume 46, Issue 5, 15 April 2004, p. 293-300.

[4] A. H. Eden, A. Yehudai, J. Y. Gil, Precise specification and automatic application of design patterns, Proceedings of the 1997 International Conference on Automated Software Engineering ASE'97, 1997.

[5] J. Fabry, T. Mens, Language-independent detection of object-oriented design patterns, Computer Languages, Systems & Structures, Volume 30, Issues 1-2, April-July 2004, p. 21-33.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns – elements of reusable object-oriented software, Addison-Wesley, Reading, MA, 1995.

[7] B. Henderson-Sellers, Object-oriented metrics: Measures of complexity, Prentice-Hall, 1996.

[8] H. Huang, S. Zhang, J. Cao, Y. Duan, A practical pattern recovery approach based on both structural and behavioral analysis, Journal of Systems and Software, Volume 75, Issues 1-2, 15 February 2005, p. 69-87.

[9] B. Huston, The effects of design pattern application on metric scores, Journal of Systems and Software, Volume 58, Issue 3, 15 September 2001, p. 261-269.

[10] D. K. Kim, R. France, S. Ghosh, A UML-based language for specifying domain-specific patterns, Journal of Visual Languages & Computing, Volume 15, Issues 3-4, June-August 2004, p. 265-289.

[11] H. Kim, C. Boldyre, A method to recover design patterns using software product metrics, 6th International Conference, ICSR-6, Austria, Lecture Notes in Computer Science 1844, 2000, p. 318-335.

[12] K. A. Kontogiannis, R. DeMori, E. Merlo, M. Galler, M. Bernstein, Pattern matching for clone and concept detection, Automated Software Engineering vol. 3, no. 1-2, July 1996, p. 77-108.

[13] G. E. Kramer, S. T. Pope, A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80, Journal of Object-Oriented Programming 1, August/September 1988, p. 26-49.

[14] C. Krämer, L. Prechelt, Design recovery by automated search for structural design patterns in object-oriented software, Proceedings of Working Conference on Reverse Engineering, Monterey, USA, IEEE CSPress, 1996.

[15] A. Lauder, S. Kent, Precise visual specification of design patterns, ECOOP'98, Lecture Notes in Computer Science 1445, 1998, p. 114-134.
[16] J. Mayrand, C. Leblanc, E. M. Merlo, Experiment on the automatic detection of function clones in a software system using metrics, Proceedings of the 1996 International Conference on Software Maintenance, 1996, p. 244.
[17] W. B. McNatt, J. M. Bieman, Coupling of design patterns: Common practices and their benefits, Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01), 2001.
[18] D. J. Ram, M. Sreekanth, Reusable integrated components of inter-related patterns for software development, Proceedings of the Seventh Asia-Pacific Software Engineering Conference (APSEC'00), p. 364-371.
[19] D. Riehle, Composite design patterns, Proceedings of OOPSLA'97, ACM, 1997, p.218-228.
[20] F. Shull, W. L. Melo, V. R. Basili, An inductive method for discovering design patterns from object-oriented software systems, Technical report, University of Maryland, Computer Science Department, College Park, MD, 20742 USA, 1996.
[21] L. Tahvildari, K. Kontogiannis, J. Mylopoulos , Quality-driven software re-engineering, Journal of Systems and Software, Volume 66, Issue 3, 15 June 2003, p. 225-239.
[22] L. Tahvildari, K. Kontogiannis, On the role of design patterns in quality-driven re-engineering, Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR'02).
[23] P. Wendorff, Assessment of design patterns during software reengineering: Lessons learned from a large commercial project, Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR'01), 2001.