Playing Roles in Design Patterns: An Empirical Descriptive and Analytic Study

Foutse Khomh[1], Yann-Gaël Guéhéneuc[1], and Giuliano Antoniol[2]
[1] Ptidej Team—DGIGL, École Polytechnique de Montréal, Québec, Canada
[2] SOCCER Lab.—DGIGL, École Polytechnique de Montréal, Québec, Canada

E-mail: {foutsekh,guehene}@iro.umontreal.ca,antoniol@ieee.org

Abstract

This work presents a descriptive and analytic study of classes playing zero, one, or two roles in six different design patterns (and combinations thereof). First, we answer three research questions showing that (1) playing roles in design patterns is not a all-or-nothing characteristic of classes and that there are significant differences among the (2) internal and (3) external characteristics of classes playing zero, one, or two roles. Second, we revisit a previous work on design patterns and changeability and show that its results were, in a great part, due to classes playing two roles. Third, we exemplify the use of the study results to provide a ranking of the occurrences of the design patterns identified in a program. The ranking allows developers to balance precision and recall as they see fit.

1 Introduction

Design patterns are proven solutions to recurrent design problems in object-oriented software design. Their design motifs [9] describe *ideal* solutions that will be either used to generate an architecture [3] or superimposed [11] on designed (or already existing) classes of a program. Consequently, classes in a program may play n roles in m motifs, with $n > 0$, $m > 0$.

Yet, to the best of our knowledge, previous works considered that classes either play *no* role or *some* role(s) in *some* motif(s), without distinguishing classes playing one or more roles in one or more motifs. They neglected that classes may play many different roles and considered role playing as a *all-or-nothing* characteristic of classes. This coarse-grained perspective prevents studying *finely* the impact of motifs on classes.

A reason for the current coarse-grained perspective is the lack of a method and manually-validated data to identify and evaluate the characteristics of classes playing one, two, or more roles with respect to classes playing no role. Therefore, we present a descriptive and analytic study of the impacts of playing one or two role(s) in motifs on classes *wrt.* playing zero role. We also show that this novel fine-grain perspective on role playing benefits works on design patterns.

We exemplify these benefits on previous works that studied (1) design motif changeability and (2) motif identification. First, Bieman [4], Di Penta [7], and others (see Section 2) showed that classes playing some role(s) in one design motif are more complex and–or change-prone than classes playing *no* roles. They did not distinguish classes playing different numbers of roles. Consequently, they could only conclude generally on role change-proneness. A fine-grain perspective allows us to revisit these previous works and show that classes playing two roles represent, in average, 56% of all the changes that occurred before the studied release date while classes playing one role only 33%.

Second, Tsantalis *et al.* [24], Guéhéneuc and Antoniol [9], and others (see Section 2) proposed approaches to identify occurrences of design motifs in programs, which return unordered sets of occurrences. Yet, a ranking would help developers focus on the most relevant occurrences first. A fine-grain perspective allows us to sketch an approach to assign ranks to occurrences in function of the numbers of roles played by their classes. Applying this approach on JHotDraw v5.1, the Decorator design motif, and the occurrences obtained from DeMIMA [9] leads to 100% precision and recall on the first occurrence, to be compared to the previously reported 7.7% precision and 100% recall.

Thus, the contributions of this paper are: (1) a descriptive study showing that a non-negligible proportions of classes play one or two roles and that some roles are often played in pairs; (2) an analytic study showing that internal and external characteristics of classes are

impacted differently by playing one and two roles; (3) a revisit of previous works confirming the soundness of our study and showing that they should be reexamined with a finer-grain perspective; and, (4) a revisit of a design pattern identification approach illustrating the ranking of occurrences and the possible improvements in precision and recall.

Section 3 presents the study definition and design while Section 4 its implementation: the method, programs, and motifs to build the samples; the metrics and their computations. Section 5 provides the study results. Section 6 present possible threats. Section 7 revisit previous works by describing two uses of the study results. Section 8 concludes with future work.

2 Related Work

Many works are related to design patterns, from their definition [15] to their identification [9]. We present here works related to the impact of design motifs on object-oriented quality and to the identification of occurrences of motifs in programs.

Motif Impacts. Bieman and McNatt [20] performed a qualitative study of the coupling between motifs and claimed that, when motifs are loosely composed and abstracted, maintainability, modularity, and reusability are well supported by the motifs. They concluded on a need for further studies to examine different motif compositions and their impact on quality.

Di Penta *et al.* [7] studied the change-proneness of roles and the kinds of changes affecting roles. Their results confirmed the expected, theoretical impact of motifs, *e.g.*, in Abstract Factory, classes playing concrete roles change more often than these playing abstract roles. They also highlighted deviations from the intuition, *e.g.*, in Composite, classes playing the role of Composite can be complex and undergo many changes.

Hannemann and Kiczales [11] studied the use of aspect-oriented programming and show that 17 of the 23 design patterns in [8] benefits from their "aspectisation" to overcome: the influence of motifs on programs and of programs on motifs; the loss of motif modularity and of traceability; the invasiveness of motifs; the difficulty to reason about classes involved in several motifs.

Khomh and Guéhéneuc [22] performed an empirical study of the impact of the 23 design patterns from [8] on ten different quality characteristics and concluded that patterns do not necessarily promote reusability, expandability, and understandability, as advocated by Gamma *et al.* They also studied patterns with respect to object-oriented principles and concluded that patterns do not necessarily lead to programs with good quality. Overall, their study advocate a considered use of patterns during development and maintenance.

Lange and Nakamura demonstrated [17] that patterns can serve as guide in program exploration and thus ease program comprehension. Through a trail of patterns, they showed that if patterns were recognized during the comprehension process, they help in "filling in the blanks" and in starting the next exploration.

Vokac *et al.* [25] analysed the corrective maintenance of a large commercial program over three years and studied the defect rates of classes playing roles in design motifs. Classes in motifs were less defect prone than others. He also noticed that the Observer and Singleton motifs are correlated with larger classes; classes playing roles in Factory Method were more compact, less coupled, and less defect prone than others classes; and, no clear tendency exists for Template Method.

Wendorff [26] evaluated the use of design patterns in a large commercial software systems and concluded that design patterns do not improve a system design necessarily. Indeed, a design can be over-engineered [16] and the cost of removing patterns high.

Wydaeghe et al. [27] studied the use of six design patterns to build an OMT editor. They discussed the impact of their motifs on reusability, modularity, flexibility, and understandability. They concluded that, although design patterns offer several advantages, not all of their motifs have a positive impact on quality.

Motif Identification. Our recent survey [9] of design motif identification approaches show that most approaches do not rank the identified occurrences. For example, Tsantalis *et al.* [24] proposed an approach based on similarity scoring to identify classes potentially playing a role in the design motif. This approach is fast and has reasonable precision and recall. It is exemplified on three programs and 10 design motifs. The occurrences are not ranked by their similarity score.

Our approach, DeMIMA [9], uses explanation-based constraint programming to provide approximations and explanations on the occurrences. It assigns a weight to each occurrence but this weight is subjective: it essentially depends on the weight assigned to each constraint and on the user's choice of the relaxed constraints; it does not consider the probability of a class to play zero, one, or more roles.

To the best of our knowledge, only Jahnke *et al.* [14] provide ranked occurrences. They used fuzzy-reasoning nets to identify design motifs. Their approach computes, for example, the probability of a class to be a Singleton. The main advantage of their approach is that fuzzy-reasoning nets deal with inconsistent and incomplete knowledge and that each occurrence is assigned a probability. However, their approach requires the description of all possible approxi-

mations of a design motif and users' assumptions.

Our study builds on this previous work, in particular Bieman and McNatt's work, to understand the impact on classes of playing one role in a motif or two roles in two different motifs. We use the study results to revisit previous works on design motif change-proneness and to rank identified occurrences of motifs. Spinellis' study [21] of four OS kernels also inspired us.

3 Study Definition and Design

Following GQM [2], the *goal* of our study is to study classes playing zero, one, or two roles in some design motifs. Our *purpose* is to bring generalisable, quantitative evidence on the impact of playing roles on classes. The *quality focus* is that playing zero, one, or two roles impact differently classes. The *perspective* is that both researchers and practitioners should be aware of the impact of playing roles on classes to make inform design and implementation choices and to understand and forecast the characteristics of classes. The *context* of our study is both development and maintenance.

3.1 Research Questions and Hypotheses

Descriptive Questions. The two first research questions are descriptive and aim at understanding the extent of classes playing zero, one, or two roles in a general population of classes. If the proportions are not negligible, then the two following analytic questions will be answered.

- **RQ1**: What is the proportion of classes playing zero, one, or two roles in some motif(s)?
- **RQ1bis**: What are the roles that are more often played solitary or in pairs than others?

Analytic Questions. The two following questions are analytic and divides in two sets of null hypotheses.

- **RQ2**: What are the internal characteristics of a class that are the most impacted by playing one or two roles *wrt.* zero role?
- **RQ3**: What are the external characteristics of a class that are the most impacted by playing one or two roles *wrt.* zero role?

For any metric m measuring some internal or external characteristics of a class, we test the set of null hypotheses: $H_{0mi/j}$: the distribution of the values of metric m for the classes playing $i \in [1,2]$ role(s) is similar to that of classes playing $j \in [0,1] \wedge i \neq j$ role.

We relate the following independent and dependent variables to assess the proportions of classes playing different roles and to test the previous null hypotheses.

3.2 Independent Variables

In an ideal situation, we would know the general population of *all possible classes* and know the number of roles played by any class. Then, we would use the sub-populations of classes playing zero, one, or more roles to answer the research questions. However, this situation is impossible because the population of all possible classes is so large and, in general, a class does not know if it plays any roles.

Therefore, the independent variables are three samples of classes playing zero, one, and two roles in design motifs. We limit our study to two roles and will consider more roles in future work. We name these samples the 0-, 1-, and 2-role samples. The samples must be large enough to be statistically representative but small enough to be manually validated, because it is not practicable for any set of classes to identify and validate manually all classes playing n-role. The method to build these samples along with its implementation are presented in Section 4.

3.3 Dependent Variables

The dependent variables are the metrics measuring classes internal and external characteristics. We choose to study a large number of metrics, as previous work [21], to assess all the possible impacts of role playing.

Internal Characteristics are related to class themselves and are measured using 56 different metrics from the literature, including Briand *et al.*'s class-method import and export coupling [5]; Chidamber and Kemerer's Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM5), and Weighted Method Count (WMC) [6]; Hitz and Montazeri 'C' connectivity of a class [12]; Lorenz and Kidd numbers of new, inherited, and overridden methods and total number of methods [18]; McCabe's Cyclomatic Complexity Metric (CC) [19]; Tegarden *et al.*'s numbers of hierarchical levels below a class and class-to-leaf depth [23]. The definitions of all the metrics is available on-line[1].

External Characteristics are limited in this study to the change-proneness of classes. A class is change-prone if, at a given time, it has been changed more than other classes. Change-proneness is assessed by computing the numbers and frequencies of past and future changes per class. Future work will study issue-proneness as well as other external characteristics.

[1] `http://wiki.ptidej.dyndns.org/research/pom`.

The computation of the internal and external characteristics is described in Section 4. In Section 5, we report the metrics that proved to be significantly impacted by the number of roles played by classes and also discuss the not-impacted metrics.

3.4 Descriptive and Analytic Analyses

We use the following analyses to answer the research question with independent and dependent variables.

RQ1. Given a population of classes from 6 programs, we computed the classes playing zero, one, and two roles with our identification approach DeMIMA. Then, we compute the accuracy of our approach for one and two roles by manually validating classes playing roles in the identified occurrences. With this precision, we extrapolate the proportions of classes playing zero, one, and two roles in the general population.

RQ2 and RQ3. We use the Wilcoxon rank-sum test to compute for each metric and each pair of samples (0-role, 1-role), (0-role, 2-roles), and (1-role, 2-role), the p-values for the corresponding null hypotheses. The Wilcoxon rank-sum test is a non-parametric statistical hypothesis test that assesses whether two samples come from a same distribution or not. It allows us to attempt rejecting the null hypotheses while making no assumptions on the normality of the samples.

4 Study Implementation

The following subsections detail the building of the samples and the computation of the metrics.

4.1 Definitions

We define a:

- **General population** as the set of all classes and interfaces belonging to some given programs;
- **n-role population** as the population of classes playing n roles in some design motifs. Thus, The *0-role* population contains all the classes in the general population playing no role. The *1-role* population contains classes playing one and only one role. The *2-role* population includes only classes playing two roles in two different motifs, *i.e.*, playing roles in *pairs of motifs*;
- **n-role class subset** as a subset of the classes in the general population that has been manually studied to identify n-role classes;
- **n-role sample** as the intersection of the n-role class subset and the n-role population: a manually validated sample of n-role classes.

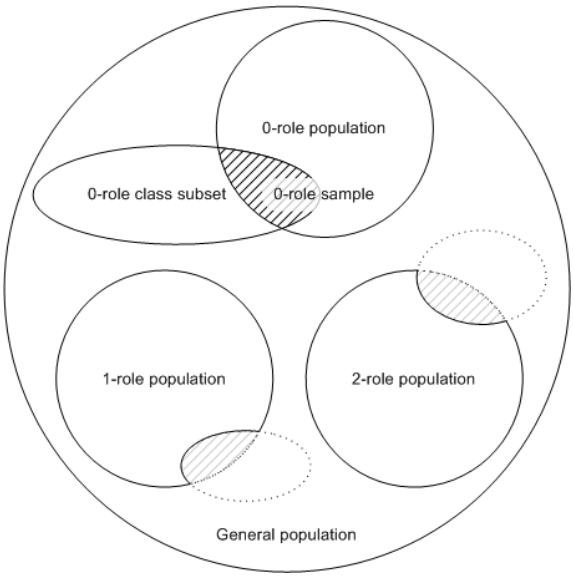


Figure 1. Subsets of the general population, details are given for 0-role classes.

Figure 1 illustrates the partition of a general population of classes. We define three sub-populations, which form a partition of the general population. The *0-role* population contains all classes playing no role. The *1-role* population contains classes playing one and only one role. The *2-role* population includes only classes playing two roles in two different motifs, *i.e.*, playing roles in *pairs of motifs*. We extracted from the 0-, 1-, and 2-role populations three subsets of classes, CS_0, CS_1, and CS_2, that we manually validated to build, after validation, the 0-, 1-, and 2-role samples with which we will answer the research questions. We use different n-role class subsets when identifying classes playing n role(s) to avoid any bias.

4.2 Size of the Samples

The size of the samples must be large enough to allow the generalisation of the results to the overall population yet small enough to be validated by hand.

We compute the sample size in two steps: (1) we assume the normality of the population and we compute the sample size needed for a two-sample t-test; and, (2) we adjust this size based on the Asymptotic Relative Efficiency (ARE) [13] of the two-sample Wilcoxon test.

We choose a *power of* 0.8, *i.e.*, we seek 80% chance of finding statistical significance if the specified effect exists. We also choose a *significance level of* 0.05 because we seek to reduce as much as possible the probability that a positive finding is due to chance alone

With this power and significance level, we study the relation between effect size and sample size to choose the adequate sample size for a two-sample t-test, assuming the normality of the distribution. We plotted the values of the sample sizes corresponding to the effect sizes varying from 0.5 to 1.5.

Figure 2 presents the obtained curve.

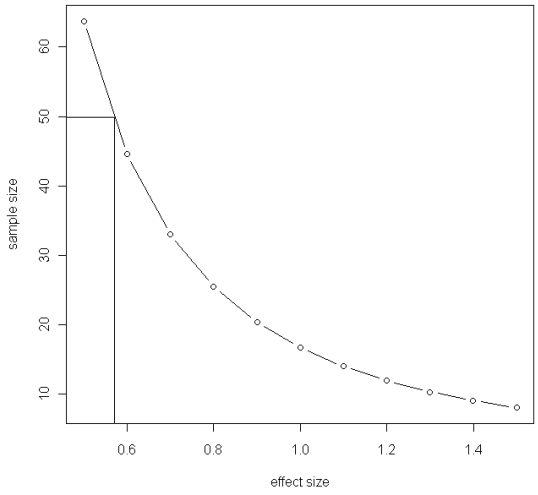


Figure 2. Possible sample sizes *wrt.* **effect size for the t-test.**

With the relations display on this curve, we decided to choose a *medium effect size of 0.58* that corresponds to a sample size of 50 classes.

The ARE represents the asymptotic limit of the ratio of the sample sizes needed to achieve equal power for two statistical tests: given a sample size for a statistical test A achieving a power p, the sample size needed for a test B to achieve the same power p is obtained from the ARE of A *wrt.* B. We compute the sample size for the two-sample Wilcoxon test that ensures the same power as the t-test, with no assumption of the distribution. The ARE for the two-sample Wilcoxon test is never less than 0.864 [13], we choose to be conservative and therefore divide the sample size for a t-test by 0.864. We obtain a *sample size of 58 classes.*

Consequently, the parameters of our study are thus:

- **Power**: 0.8;
- **Significance level**: 0.05;
- **Effect size**: 0.58;
- **Sizes of the samples**: 58 classes.

4.3 Selection of the General Population

We choose six programs to form the general population of classes from which to build the n-role samples: ArgoUML v0.18.1, Azureus v2.1.0.0, Eclipse JDT Core plug-in v2.1.2 (JDT Core v2.1.2), JHotDraw v5.4b2, Xalan v2.7.0, and Xerces v1.4.4. These programs are written in Java and open source. They are of different domains, sizes, complexity, and maturity. Table 1(a) summarises facts on these programs.

ArgoUML [2] is a full-fledged UML modelling tool with code generation and reverse-engineering capabilities. It provides the user with a set of views and tools to model programs using UML diagrams, to generate the corresponding code skeletons and to reverse-engineer diagrams from existing code. Azureus [2] (now called Vuze) is a bit-torrent client. Bit torrent is a protocol to exchange data among peers across a network. Azureus provides advanced user-interface and implementation of the protocol. JDT Core is an Eclipse [2] plug-in that implements the infrastructure for the Java IDE of the Eclipse platform. It provides a Java model and capabilities to parse, manipulate, and rewrite Java programs. JHotDraw [2] is a graphic framework for drawing 2D graphics. It was created in October 2000 by Beck and Gamma with the purpose of illustrating the use of design patterns. Xalan [2] is an XSLT processor for transforming XML documents into other document types (HTML, text, and so on). It implements the XSLT and XPath standards. Xerces [2] is a Java XML parser which supports XML, DOM, and SAX.

4.4 Selection of the Motifs and their Roles

We select six design motifs used in previous work [7, 24]: Command, Composite, Decorator, Observer, Singleton, and State. We follow [7] in their choice of the motifs *main* roles. We only study main roles because (1) they are most likely to impact classes, as confirmed by the following results, and (2) they allow us to concentrate on a fewer number of roles during the manual validation. In the following, roles are named using the notation <Pattern Name> . <Role Name>.

[2] http://argouml.tigris.org/, http://azureus.sourceforge.net/, http://www.eclipse.org, http://www.jhotdraw.org, http://xml.apache.org/xalan-j/, and http://xerces.apache.org/xerces-j.

Programs	NOC	LOC	Release Dates	Past Changes	Future Changes	Issues
ArgoUML v0.18.1	1,267	202,520	30/04/05	20,290	12,617	41,565
Azureus v2.1.0.0	591	83,534	1/06/04	18,304	483	33,753
JDT Core v2.1.2	669	184,690	3/11/03	23,243	26,923	62,728
JHotDraw v5.4b2	413	44,898	1/02/04	5,793	51	1,286
Xalan v2.7.0	734	259,286	8/08/05	12,298	1,714	58,448
Xerces v1.4.4	306	86,814	13/10/03	5,213	1,209	16,143
Total	3,980	861,742	6 releases	85,141	42,997	213,923

(a) Statistics for the six programs. (Future refers to the time between the release dates and 31/01/09.)

Programs	Expected	Zero Role	One Role	Two Roles
ArgoUML v0.18.1	17	17	10	10
Azureus v2.1.0.0	9	9	9	9
JDT Core v2.1.2	10	10	17	17
JHotDraw v5.4b2	6	6	6	6
Xalan v2.7.0	11	11	11	11
Xerces v1.4.4	5	5	5	5
Total	58	58	58	58

(b) Distribution of the sample size among the programs of our strata.

Table 1. Data on the Studied Programs.

Patterns	Descriptions	Main Roles
Command	Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations	Command, Invoker
Composite	Composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly	Component, Composite
Decorator	Attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality	Component, Decorator
Observer	Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically	Observer, Subject
Singleton	Defines a mechanism that ensure that the same instance of a class is used throughout a program execution	Singleton
State	Allows an object to alter its behavior when its internal state changes	Context, State

Table 2. Chosen design patterns and the main roles of their motifs.

In addition to choosing the roles of interest, we must also select pairs of roles for classes playing two roles. The eleven main roles yield 66 possible pairs of roles. We exclude pairs with the same role because identical roles in different motifs must have similar characteristics, *e.g.*, among the six motifs, Component is the only role that appears twice with similar structure albeit slightly different semantics. We exclude pairs involving roles from the same motif because a class playing both the roles of Composite.Component and Composite.Composite must be a degenerated case. Consequently, we retain 45 possible pairs.

4.5 Building of the Samples

Building the n-role sample, with $n \in [0, 2]$, consists of searching in the general population for three sets of 58 classes playing n roles. We reduce the search space using our DeMIMA approach because it ensures 100% recall and has up to 80% of precision, with an average of 40% for the six design motifs in Table 2 in a set of programs different from these used in this study.

DeMIMA uses explanation-based constraint programming to automatically provide (1) explanations on the identified occurrences: the roles and relationships that led to identify a certain micro-architecture as an occurrence of a motif and (2) approximations from the given motifs: the relaxations of the constraints on a micro-architecture to be identified as a approximated occurrence of a motif. It provides a complete mapping between roles in a motif and classes in an occurrence. Thus, we can find all the roles that each class plays for a set of motifs in a program.

We applied DeMIMA on the classes in the general population and obtain candidate classes playing (at least) one role in the selected motifs. We automatically divided this set in two 1- and 2-role subsets.

Then, for each subset, we studied each class (its code source, comments, hierarchy, relationships) to decide whether it plays one role (respectively two roles) using a voting process: the authors and a post-doc. student marked independently each class as *true* when a class played one role (respectively, two roles) or $false$ else. Then, a class was assigned to the 1-role sample (respectively, 2-role sample) if the majority marked it as *true*, else it was excluded. We stopped the voting process as soon as the samples were completed.

In total, 238 classes were manually validated: 81 classes where false positives, *i.e.*, classes playing no role but belonging to occurrences identified by DeMIMA; 88 classes played 1 role; and, 69 classes played 2 roles. Finally, from the classes *not* included in any of the oc-

currences identified by DeMIMA, we selected randomly and validated manually 58 classes playing 0 role.

The distribution in the samples of the classes from the general population must be representative of the population. We distributed the 58 classes per sample along the strata formed by the six programs. We computed stratified sample sizes so that each stratum reflected the proportional size of one program with respect to the others. For example, JHotDraw v5.4b2 makes up 10.38% of the general population. So, it had to provide 10.38% of the 58 classes in each sample. Thus, we ensured that the results equally reflect the six programs. The second column in Table 1(b) shows the expected size of each stratum, *i.e.*, the expected numbers of classes of each program in each sample.

We could not find enough 1- and 2-role classes in ArgoUML. Therefore, we made up for the reduced number of classes in ArgoUML by using more classes from JDT Core. The fourth and fifth columns in Table 1(b) show the actual repartitions of classes in the 1- and 2-role samples. We replicated our study on the general population without JDT Core and on JDT Core exclusively and noticed the same trends.

4.6 Computing Dependent Variables

We compute the dependent variables using two different frameworks.

Internal Characteristics are computed using the PADL meta-model and parsers and the POM framework [10]. PADL models of programs are obtained using the Java parser and the metric values are computed by applying each metric on each class of the models.

External Characteristics are computed using the Ibdoos framework. Ibdoos extracts commit information from any CVS, GIT, or SVN repository and stores this in a database. We implemented queries to count the numbers and frequencies of changes for each class before and after the release dates of the six programs. Issue-proneness is assessed by analysing the issue-trackers of each program and counting the number of classes in some issue descriptions.

5 Study Results

We analyse the metrics values computed on the classes in the samples to answer the research questions.

RQ1. To answer our first research question, "What is the proportion of classes playing zero, one, or two roles in some motifs in a program?", we extrapolate, for each program and each motif, the number of classes playing zero, one role, and two roles in the motifs.

Programs	Candidates	One Role	Two Roles
ArgoUML v0.18.1	21	3	10
		14.28%	47.61%
Azureus v2.1.0.0	64	22	19
		34.37%	29.68%
JDT Core v2.1.2	67	30	22
		44.77%	32.83%
JHotDraw v5.4b2	30	11	13
		36.66%	43.33%
Xalan v2.7.0	55	23	11
		41.81%	20.00%
Xerces v1.4.4	29	10	11
		34.48%	37.93%
Total	266	99	86
		37.21%	32.33%

Table 3. Validated precisions of DeMIMA.

Programs	Total	Zero Role	One Role	Two Roles
ArgoUML v0.18.1	1,267	900	51	316
	100%	71.03%	4.02%	24.94%
Azureus v2.1.0.0	591	449	67	75
	100%	75.97%	11.33%	12.69%
JDT Core v2.1.2	669	445	46	178
	100%	66.51%	6.88%	26.60%
JHotDraw v5.4b2	413	288	24	101
	100%	69.73%	5.81%	24.45%
Xalan v2.7.0	734	594	36	104
	100%	80.92%	4.90%	14.16%
Xerces v1.4.4	306	156	94	56
	100%	50.98%	30.72%	18.30%
Total	3,980	2,832	318	830
	100%	71.15%	7.99%	20.85%

Table 4. Extrapolated numbers and percentages of classes playing no, one, or two roles.

First, from the class subsets, we compute the accuracy of DeMIMA as the number of classes in a subset *indeed* playing zero, one, or two roles with respect to the total numbers of classes in the subsets. Table 3 summarises this accuracy and shows that it varies across motifs *and* programs. It therefore highlights the need for more detailed studies of the accuracy of identification approaches. Indeed, current approaches report their precision and recall in function of the motifs but of the programs. Reporting variations in terms of programs could help the community to focus on programs in which the identification is difficult. Such a focus would lead to a better understanding of the impact of program design and implementation on analysis tools and to a collection of known difficult programs. This collection could be included into oracles for design motif identification, such as P-MARt [10].

Second, we extrapolate in Table 4 the numbers of classes playing one and two roles from the previous accuracy and the numbers of classes in each program.

The number of classes playing zero role is computed by subtracting the two previous numbers from the total number of classes. Table 4 shows that the percentage of classes playing one or two roles in any of the six selected design motif varies from 4.02% to 30.72%.

The answer to RQ1 is that classes playing one or two roles do *exist in programs and are not negligible, which confirms the need to understand the characteristics of classes playing different numbers of roles.*

RQ1bis. An answer to this research question, "What are the roles that are more often played solitary or in pairs than others?" is obtained by studying the proportions of the numbers of classes playing one or two roles with respect to the number of classes playing a particular role.

Table 5 shows the numbers and proportions for each role and each pair for which the proportions of classes playing this role or pair of roles was not negligible. It shows that, for example, five classes play the role of Command.Command alone while four play the roles of both Command.Command and State.State, which make up for 44.44% of classes playing both roles with respect to the $5 + 4$ classes playing the Command.Command role.

We notice three facts:

1. There are three pairs and two solitary roles for which the percentage is above our decision threshold: (Command.Invoker, State.State), (Decorator.Component, State.State), (Decorator.Decorator, State.State), Composite.Composite, State.Context. We could conclude that these pairs are prevalent. Yet, these roles and pairs are only played by a few classes. We will therefore study in future work more classes playing these roles to generalise the results to any programs.

2. Among the roles/pairs with a significant number of classes (more than 50) playing these roles, the percentages are smaller than our decision threshold and prevents us to generalise our results.

3. The preponderance of pairs involving roles in the State motif possibly indicates a bias during the manual validation of the classes. We further discuss the threat to the validity of our study in Section 6.

We therefore answer that, in the six studied programs, pairs (Command.Invoker, State.State), (Decorator.Component, State.State), (Decorator.Decorator, State.State) and roles Composite.Composite and State.Context have greater prevalence than others., which confirms that some roles are more often played together than others.

Pairs
(Command.Command , State.State)
(Command.Invoker , State.State)
(Command.Invoker , Singleton.Singleton)
(Composite.Component, Observer.Observer)
(Composite.Component, State.State)
(Composite.Component, Singleton.Singleton)
(Composite.Composite , State.Context)
(Composite.Composite , Singleton.Singleton)
(Decorator.Component , State.Context)
(Decorator.Component , State.State)
(Decorator.Component , Singleton.Singleton)
(Decorator.Decorator , State.State)
(Observer.Observer , State.State)
(Singleton.Singleton , State.State)

Table 6. Selected pairs of roles.

RQ2. In the rest of this study, we limit our selection of roles to the 14 pairs shown in Table 6 by keeping only pairs for which we had enough classes, determined while in RQ1. To answer RQ2, "What are the internal characteristics of a class that are the most impacted by playing one or two roles?", we test the null hypotheses $H_{0mi/j}, i \in [1,2], j \in [0,1] \wedge j \neq i$ for the 56 metrics. Table 8 summarises the results. It shows for each metric in each metric group the p-value when testing the associated null hypothesis. It reports in bold the p-values that show a statistically significant difference between the distribution of the metric value between two samples. It also shows using arrows the trend in the change between.

We analyze the results in Table 8 in three steps: metrics whose distributions do not change between samples and then each pairs of samples:

There are 8 metrics whose distributions did not change significantly between the three samples: ANA, connectivity, CP, DSC, MFA, NOH, PP, and RPII. These metrics are therefore unlikely to be of interest when assessing the impact of role playing and could be excluded from future studies on design motifs. This finding was predictable for CP, PP, RPII because these metrics measure the structure of the packages of a system rather than the structure of its classes. The same explanation applies to DSC and NOH, which count respectively the total number of classes and the number of class hierarchies in a system. The finding for ANA, connectivity, and MFA is surprising because we expected that classes playing roles in design motifs would inherit more from other classes and are more "connected" to other classes. We explain this finding by the specific definitions of these three metrics because the values of other metrics related to inheritance and

Roles	Pairs	Counts	Percentages
Command.Command		5	55.56%
	(Command.Command , State.State)	4	44.44%
Command.Invoker		0	0%
	(Command.Invoker , State.State)	13	**100%**
Composite.Component		4	22.22%
	(Composite.Component, Observer.Observer)	9	50%
	(Composite.Component, State.State)	5	27.78%
Composite.Composite		8	**80%**
	(Composite.Composite , State.Context)	2	20%
Decorator.Component		1	11.11%
	(Decorator.Component , State.State)	8	**88.89%**
	(Decorator.Component , State.Context)	0	0%
Decorator.Decorator		1	8.33%
	(Decorator.Decorator , State.State)	11	**91.67%**
Observer.Observer		36	66.67%
	(Composite.Component, Observer.Observer)	9	16.67%
	(Observer.Observer , State.State)	9	16.67%
State.Context		33	**94.29%**
	(Composite.Composite , State.Context)	2	5.71%
	(Decorator.Component , State.Context)	0	0%
State.State		**50**	**50%**
	(Command.Command , State.State)	4	4%
	(Command.Invoker , State.State)	13	13%
	(Composite.Component, State.State)	5	5%
	(Decorator.Decorator , State.State)	11	11%
	(Decorator.Component , State.State)	8	8%
	(Observer.Observer , State.State)	9	9%

Table 5. Counts and percentages of roles played alone or paired with another role.

coupling significantly change between the samples.

There is a statistically significant difference between classes playing zero and one role for 29 metrics. These metrics characterise coupling, cohesion, inheritance, size and polymorphism, and complexity. The trends are a decrease in metric values for only four metrics: LCOM1, LCOM2, WMC1, and RRTP. This finding is explained again by the implementations of the metrics: LCOM1 and 2 have been superseded by LCOM5, which changes significantly, while WMC1 counts weighs each method by 1 and RRTP is related to packages. The others metrics see a statistically significant increase in their values. Among these, we can quote: CBO, DCAEC, LCOM5, McCabe, SIX, WMC. We explain this finding by the fact that playing roles implies responsibilities, thus classes playing one role have more responsibilities than classes playing zero role, which results in classes being more complex (McCabe, SIX, WMC), more coupled (CBO, DCAEC), and less cohesive (LCOM5), as examples. We conclude that playing one role impact classes *wrt.* playing zero role.

There is a statistically significant difference between classes playing zero and two roles for 48 metrics, with, for each metric, an increase of its values for classes playing two roles, except for RRFP and RRTP. This finding was expected because RRFP and RRTP concern packages. For the 48 other metrics, the argument of added responsibilities with each role can also help explain the impact of 2-role classes on metric values in comparison to the impact of classes playing zero role. Having more responsibilities, classes become more complex (McCabe, WMC, WMC1, SIX), more coupled (CBO, DCAEC, DCC, DCMEC), inherit more from their superclasses (CLD, DIT, NOC, NOD), and use more polymorphism (MOA, NMA, NMD). Therefore, we conclude that playing two roles has a major impact on classes, in particular in comparison to the impact of playing one role. Playing two roles should be carefully considered during design and implementation.

The change in the distributions of the metrics values between classes in the 2- and 1-role samples is significant for 26 metrics, among which: CAM, CLD, DCC, LCOM5, McCabe, SIX, WMC. We observe that the more they play roles, the more classes are complex (McCabe, SIX, WMC, WMC1), are coupled (CBO, DCC), inherit (NOP), and use polymorphism (MOA, NAD, NMO). The values of CLD decrease significantly, possibly hinting at more shallow inheritance tree thank to the elegant solutions provided by the motifs. We conclude that, indeed, playing two roles has a significant impact on classes that cannot be accounted for by the fact that they play two different one roles.

Table 7 shows the change in trends in the metrics values when considering classes playing zero roles but identified by DeMIMA as playing one or more roles, *i.e.*, false positives, rather than classes truly playing zero roles. It shows that, again, there are several metrics that are impacted when comparing classes in the 0^{FP}-sample with classes in the 1- and 2-roles samples.

Consequently, the answer to RQ2 is that playing two roles has a major impact on classes when compared to playing zero or one role.

RQ3. We answer the last research question, "What are the external characteristics of a class that are the most impacted by playing one or two roles?", by carrying null hypothesis tests on the numbers and frequencies of past and future changes and on the numbers of issues related to classes in the different samples. Table 8 shows the results of testing out the null hypotheses.

We can reject the null hypotheses related to the external metrics for 1-role and 2-role classes *wrt.* 0-role classes with statistical significance. We cannot reject the null hypotheses for 2-role classes when compared to 1-role classes.

These results confirm previous works on the change- and issue-proneness of classes playing roles in some design motifs, for example [4, 7]. We perform in Section 7 a deeper analysis that shows that 2-role classes are the cause of the greater parts of the changes (56%) and issues (57%) with 1-role classes causing only 33% of changes and 30% of issues.

The answer to RQ3 is that playing roles do impact the number of changes and issues as well as the frequencies of the changes. It confirms that playing roles has a major impact on change- and issue-proneness.

6 Threats to Validity

The results of any empirical studies are subject to the following threats to their validity. **Construct Validity.** There is actually no agreed-upon definition of motif composition. In this study, we define a motif composition as the implementation of two different roles in two different motifs by a same class. We only considered pairs of roles and ignored the effect of the particular roles on a class. We also explicitly excluded auto-composition, *i.e.*, a class playing two different roles in a same motif. Future work should distinguish compositions based on their roles and further study auto-compositions. Also, we purposefully studied only main roles of design patterns. Future work includes extending our study to all roles.

Internal Validity. Our approach relies on the precision of the automatic detection technique DeMIMA. The results include false positive. We try to limit the number of false positive through a manual validation. However, the manual validation is a tedious task that leads to resilience and the experimenter bias: some false positive class may pass the validation because it "looks like" a motif. An approach that would provide a better precision is to use a manually validated repository of motifs such as P-MARt [10]. However, P-MARt does not contain enough data as of now to perform such a study. We used as a baseline for our study of classes playing 1-role and 2-role, the 0-role population of classes playing none of the 11 roles considered in our study. However, among these classes, some may be playing one or two roles in *other* design motifs. Future work should extend this study to cover the 23 patterns from Gamma *et al.* [8]

External Validity. We studied six programs of different sizes, domains, maturity, and complexity. However, these programs are all open-source programs written in Java. We choose six design patterns among the many available. The results could be different with industrial programs, other object-oriented programming languages, and different design patterns.

Reliability validity. This threat concern the possibility of replicating this study. We attempted to provide all the necessary details to replicate our study. Moreover, both Eclipse source code repository and issue-tracking system are available to obtain the same data. Finally, the data from which our statistics have been computed is available on-line[3].

Statistical Validity. In Section 4, we presented the process to build the sample size of our study. We could not find enough classes playing one role and two roles in ArgoUML and, therefore, used more classes from JDT Core. We assess the impact of this selection on the conclusions of our study by replicating the study on the population without JDT Core and on JDT Core exclusively. We obtained for these two additional studies the same trends on the results.

Conclusion Validity. There is no threat to the validity of the conclusion of this study as there is a direct relation between the chosen metrics and the overall internal quality of a class.

7 Discussions

We now exemplify the use of our study results by revisiting previous works and sketching an approach to rank occurrences of identified design motifs.

7.1 Proportions of 0-, 1-, or 2-role Classes

Table 4 shows the percentages of classes playing no, one, or two roles in the six programs. In addition to the overall percentages, some programs have higher percentages than others: JHoDraw contains *only* 5.81% of classes playing one role and 24.45% two roles in contrast to the 30.72% of classes playing one role in Xerces and the 26.60% of classes playing two roles in JDT Core. Given that JHotDraw has been developed to show the "good" use of design patterns, the higher

[3] `http://www.ptidej.net/downloads/experiments/prop-icsm09/`.

percentages of classes playing one or two roles in Xerces and JDT Core could be due to an *overuse* of design patterns. These higher percentages could be used with other quality measures to confirm or refute the impact of overusing design patterns as put forward by Wendorff [26] and others.

7.2 Trends in Playing Roles on Quality

Table 8 shows that in the 2-role sample, classes are more complex, more coupled, less cohesive than classes than in the 0- and 1-role samples. This trend suggests that motif composition (playing more than one role in motifs) degrades more the quality of the classes. We explain this degradation by the addition to the classes of non-feature oriented methods and fields to allow the classes to fulfill their roles. Future work should investigate classes playing more roles in motif compositions to confirm and generalise this trend.

7.3 Revisit of Previous Works.

Bieman and McNatt's Work. We observe that playing one or more roles in a design motif decreases the cohesion of classes (increases of the LCOM$\star$ metrics) while increasing their coupling (increase of the coupling metrics). This result confirm Bieman and McNatt's claim [20] that design motifs impact the cohesion and coupling of programs.

Hannemann and Kicazles' Work. We explain the decrease in cohesion and increase in coupling by suggesting that design motif-related methods may be orthogonal to the responsibilities of the classes and thus reduce their cohesion. Therefore, our study confirms that design motifs are often "cross-cutting concern" that could benefit from being "separated" from the program using, for example, aspect-oriented programming. We thus bring quantitative support to previous work on rewriting design motifs as aspects [11].

Di Penta *et al.*'s Work. We revisit Di Penta *et al.*'s study of the numbers and frequencies of changes of classes playing roles. We compare the set of classes playing some roles, as identified by DeMIMA, which is the union of the samples of 1- and 2-role classes with the sample of false positive classes, noted 0^{FP}, with the set of classes playing *really* zero role: 0-role sample vs. (0^{FP}-role $\cup$ 1-role $\cup$ 2-role) sample. This comparison yields a p-value of **1.973e-14** < 0.05, thus confirming the previous work as well as the statistical validity of our three samples.

It appears from our study that, in average, the numbers of changes prior to the releases of the studied program for classes playing two roles accounts for 56% of the total number of past changes. Also, classes playing two roles change 1.52 times more than classes playing one role. Classes playing zero and one role account respectively for 33% and 11% of past changes. Classes playing one role change more than two role classes after the studied release of the programs. They change 1.46 times more than the 2-role classes and they account for 61.53% of the total number of future changes. We explain this result by the fewer numbers of future changes, shown in Table 1(a): in total, there are twice as much past changes than future changes. Therefore, we bring evidence that the results found by Di Penta *et al.* was largely due to classes playing two roles.

Table 7 shows the fine-grained study of the impact for a class to be a false positive *wrt.* playing zero, one, or two roles. It shows that false positive classes do have a significantly different number of changes than classes playing 0 role. This results was expected because false positive classes must have some particular feature, because DeMIMA included them in its results. Also, it shows that classes playing two roles change significantly differently from false positives classes, thus confirming their importance.

We conclude that developers should be careful with classes playing roles, in particular 2-role classes, because they have internal and external metric values that are significantly higher than these of other classes: they are more change-prone, less cohesive, more coupled, more complex, and more issue-prone.

7.4 Ranking Design Motif Occurrences.

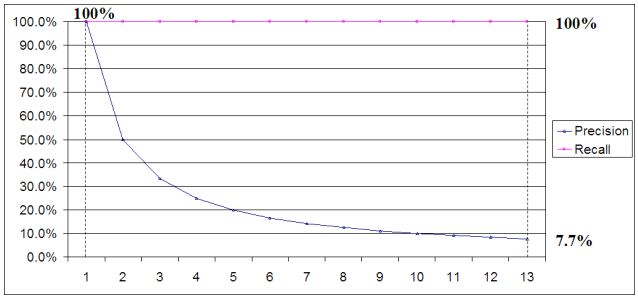
We get inspiration from previous works by Antoniol *et al.* [1], Guéhéneuc *et al.* [10], and Jahnke *et al.* [14] to use the study results to rank occurrences.

First, we assign to each class in a program its probability to play one or more roles in a design motif using its metrics values. We select a set of discriminating metrics for the 0-, 1- and 2-role classes from Table 8. Then, we plot the distributions of these metrics for the 0-, 1- and 2-role samples. Finally, we find the thresholds characterising these samples for each selected metrics by superposing the curves of each selected metrics.

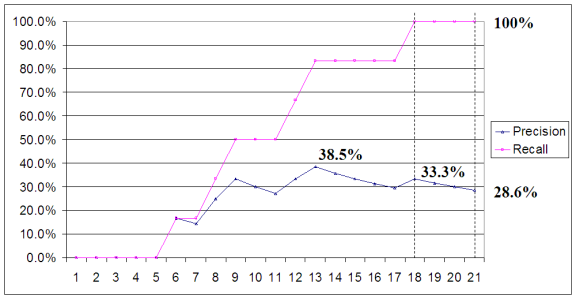
Second, the probability of a class is computed by interpolation as the distance between the values of its metrics and the thresholds characterising each samples. We aggregate these probabilities with the min and max fuzzy logic operators. Finally, from the probability of classes, we assign a probability to an occurrence as:

$$p_O = \frac{\sum_{i=1}^{n} \alpha_i \times p_{C_i}}{\sum_{i=1}^{n} \alpha_i}$$

where p_O is the probability of the occurrence to be a



(a) Precision and recall of the identification of Decorator.



(b) Precision and recall of the identification of State.

Figure 3. Precision and Recall.

true positive; p_{C_i} is the probability of the class playing the i^{th} role in the occurrence to play one or more roles; and, α_i is a weight to discriminate roles.

We apply this naive approach using the metrics CAM, CBO, LCOM5, McCabe, MOA, NAD, NMO, SIX, and WMC, because Table 8 shows that these metrics are the most discriminating of classes playing 0, 1, and 2 roles. We choose $\forall i \in [1, n], \alpha_i = 1$. We apply this approach on the occurrences identified by DeMIMA in JHotDraw v5.1. We choose JHotDraw v5.1 to be able to compare with our previous work [9] ans also to show that our naive approach can be applied successfully on a different set of programs.

Figure 3(a) shows that, in the case of Decorator, our naive approach assigns the higher rank to the true positive occurrence. The precision and recall are therefore 100% when considering the first occurrence. These are to be contrasted to the 7.7% precision and 100% recall obtained by DeMIMA with no ranking [9].

Figure 3(b) shows that, in the case of State (or Strategy), our approach rank occurrences with less efficiency. Still, the precision of 33.3% with 100% recall obtained on the 18^{th} occurrence must be compared to the DeMIMA precision of 28.6%. Also, if a recall of 100% is not mandatory, precision reaches 38.5% on the 13^{th} occurrence.

We obtain results for the other four motifs in-between those presented for Decorator and State. Therefore, this naive approach allows reducing the developers' efforts by presenting true positive occurrences first and modulating precision and recall.

We conclude that our study results allow ranking the occurrences obtained from a design pattern identification approach using the number of roles likely to be played by classes. This ranking reduces the developers' efforts and allows developers to balance precision and recall as they see fit.

8 Conclusion

In this paper, we presented a study of the impact of playing one or two roles in a(some) motif(s) for a class. We answered the following research questions: **RQ1.** In average, 8.24% (respectively 17.81%) of the classes of the six studied programs played one role (respectively two roles) in some motifs. These percentages are not negligible and therefore justify *a posteriori* the interest in design motif identification and *a priori* future studies on the impact of motifs on programs. **RQ1bis.** Despite the few numbers of classes displaying a relationships between roles, we can conclude that some roles are more often played in pairs than others, for example (Decorator.Decorator, State.State). Further studies must focus on this question to bring further generalisable evidence. **RQ2.** There is a significant increase in many metric values, in particular for classes playing two roles. These increases confirm *a posteriori* the warning addressed to the community by Bieman, Beck, and others on the use of design patterns. **RQ3.** There is a significant increase in the frequencies and numbers of changes of classes playing two roles. We thus confirmed on new samples the previous results by Di Penta *et al.*

We justify the usefulness of this study by revisiting previous work and proposing a naive approach to rank occurrences. We show that developers should be wary of classes playing two roles because they have significantly higher metric values and represent 56% of changes while 1-role classes only 33%.

We also sketched a naive approach to illustrate the possibility of ranking occurrences using the metrics characterising 1- and 2-role classes. This approach leads to a precision and recall of 100% for the first occurrence of the Decorator. Extending on this naive approach, a new family of design pattern identification approaches could be designed to include the knowledge of the numbers of roles played by classes.

As future work, we plan to further study the impact of unique design motif on metrics values with the intuition that some motifs actually *do* fulfill (part of) the intrinsic responsibilities of classes. We will also replicate this study on other motifs and programs as well as study classes playing three roles and more to confirm its generalisability. We also plan to further study the ranking of occurrences using other a more sophisticated approach, other identification approaches, and other programs. Also, the use of Bayesian beliefs networks to assign probabilities presents a great potential of obtaining better ranking and thus improving further the precision of identification approaches.

Acknowledgements. We thank Simon Denier for fruitful discussions and his help extracting the data. This work has been partly funded by Égide Lavoisier (France) and the NSERC and CFI (Canada).

9 Appendix

References

[1] G. Antoniol, R. Fiutem, and L. Cristoforetti. Design pattern recovery in object-oriented software. In *Proceedings of the* 6^{th} *International Workshop on Program Comprehension*, pages 153–160. IEEE Computer Society Press, June 1998.

[2] R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. In *IEEE Transactions on Software Engineering*, 10(6):728–738, November 1984.

[3] K. Beck and R. E. Johnson. Patterns generate architectures. In *Proceedings of* 8^{th} *European Conference for Object-Oriented Programming*, pages 139–149. Springer-Verlag, July 1994.

[4] J. M. Bieman, D. Jain, and H. J. Yang. OO design patterns, design structure, and program changes: An industrial case study. In *Proceedings of the International Conference on Software Maintenance*, pages 580–589, http://www.dsi.unifi.it/icsm2001/, November 2001. IEEE Computer Society.

[5] L. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for C++. In *Proceedings of the* 19^{th} *International Conference on Software Engineering*, pages 412–421. ACM Press, May 1997.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. Technical Report E53-315, MIT Sloan School of Management, December 1993.

[7] M. Di Penta, Luigi Cerulo, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study of the relationships between design pattern roles and class change proneness. In *Proceedings of the* 24^{th} *International Conference on Software Maintenance (ICSM)*. IEEE Computer Society Press, September–October 2008. 10 pages.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1^{st} edition, 1994.

[9] Y.-G. Guéhéneuc and G. Antoniol. DeMIMA: A multi-layered framework for design pattern identification. In *Transactions on Software Engineering (TSE)*, 34(5):667–684, September 2008. 18 pages.

[10] Y.-G. Guéhéneuc, H. Sahraoui, and Farouk Zaidi. Fingerprinting design patterns. In *Proceedings of the* 11^{th} *Working Conference on Reverse Engineering (WCRE)*, pages 172–181. IEEE Computer Society Press, November 2004. 10 pages.

[11] J. Hannemann and G. Kiczales. Design pattern implementation in Java and AspectJ. In *Proceedings of the* 17^{th} *Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 161–173. ACM Press, November 2002.

[12] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proceedings of the* 3^{rd} *International Symposium on Applied Corporate Computing*, pages 25–27. Texas A & M University, October 1995.

[13] M. Hollander and D. A. Wolfe. *Nonparametric Statistical Methods*. John Wiley and Sons, inc., 2nd edition, 1999.

[14] J. H. Jahnke and A. Zündorf. Rewriting poor design patterns by good design patterns. In *Proceedings the* 1^{st} *ESEC/FSE workshop on Object-Oriented Reengineering*. Distributed Systems Group, Technical University of Vienna, September 1997. TUV-1841-97-10.

[15] H. Kampffmeyer and S. Zschaler. Finding the pattern you need: The design pattern intent ontology. In *Proceedings of the* 10^{th} *International Conference on Model Driven Engineering Languages and Systems*, pages 211–225. Springer, September–October 2007.

[16] J. Kerievsky. *Refactoring to Patterns*. Addison-Wesley, 1st edition, August 2004.

[17] D. B. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. In *Proceedings of the* 10^{th} *annual conference on Object-oriented programming systems, languages, and applications*, pages 342 – 357. ACM Press, 1995.

[18] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics: A Practical Approach*. Prentice-Hall, 1^{st} edition, July 1994.

[19] T. J. McCabe and C. W. Butler. Design complexity measurement and testing. In *Communications of the ACM*, 32(12):1415–1425, December 1989.

[20] W. B. McNatt and J. M. Bieman. Coupling of design patterns: Common practices and their benefits. In *Proceedings of the* 25^{th} *Computer Software and Applications Conference*, pages 574–579. IEEE Computer Society Press, October 2001.

[21] D. Spinellis. A tale of four kernels. In *Proceedings of the* 30^{th} *International Conference on Software Engineering*, pages 381–390. ACM Press, May 2008.

[22] Foutse Khomh and Y.-G. Guéhéneuc. Do design patterns impact software quality positively? In *Proceedings of the* 12^{th} *Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society Press, April 2008. Short Paper. 5 pages.

[23] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi. A software complexity model of object-oriented systems. In *Decision Support Systems*, 13(3–4):241–262, March 1995.

[24] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis. Design pattern detection using similarity scoring. In *Transactions on Software Engineering*, 32(11), November 2006.

[25] M. Vokác. Defect frequency and design patterns: An empirical study of industrial code. In *Transactions on Software Engineering*, 30(12):904–917, 2004.

[26] P. Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In *Proceedings of* 5^{th} *Conference on Software Maintenance and Reengineering*, pages 77–84. IEEE Computer Society Press, March 2001.

[27] B. Wydaeghe, K. Verschaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers. Building an OMT-editor using design patterns: An experience report. 1998.

Metric Groups	Metric Names	1 role vs. 0 role		2 role vs. 0 role		2 role vs. 1 role	
		p-values	Trends	p-values	Trends	p-values	Trends
Coupling	ACMIC	0.896		0.066		**0.04961**	↗
	ACAIC	0.6251		0.8771		0.5029	
	CBO	0.513		**0.000176**	↗	**0.001948**	↗
	CBOin	0.8466		**0.0001986**	↗	**0.0005939**	↗
	CBOout	0.6496		**9.21E-06**	↗	**0.0001025**	↗
	connectivity	0.1912		0.8574		0.2603	
	CP	0.4471		**0.03687**	↗	0.1428	
	DCAEC	**0.002435**	↗	0.2118		0.06724	
	DCC	0.3617		**2.05E-05**	↗	**0.002347**	↗
	DCMEC	0.06408		0.2239		0.595	
	RFP	0.9383		0.6465		0.6074	
	RRFP	0.6811		0.993		0.5106	
	RRTP	0.8693		0.6973		0.6952	
	RTP	0.8923		0.4358		0.3693	
Cohesion	CAM	0.8532		**0.0007399**	↗	**0.0003884**	↗
	CohesionAttributes	0.5716		**0.01112**	↗	**0.0009488**	↗
	LCOM1	0.6737		**0.0004046**	↗	**0.0009946**	↗
	LCOM2	0.9168		**0.001582**	↗	**0.0017**	↗
	LCOM5	0.6976		**0.0083**	↗	**0.001383**	↗
Inheritence	AID	0.6621		**0.03946**	↗	0.1391	
	ANA	0.1938		0.5803		0.3918	
	CLD	**0.002887**	↗	0.9954		**0.003298**	↘
	DIT	0.3		**0.02209**	↗	0.2632	
	NCM	0.6426		**0.008635**	↗	0.07486	
	NOA	0.9256		**0.01207**	↗	**0.01153**	↗
	NOC	**0.003547**	↗	0.1792		0.245	
	NOD	**0.0002031**	↗	0.166		0.07	
	NOH	0.7356		0.7807		0.9663	
	NOP	0.4834		**0.0008245**	↗	**0.007146**	↗
	ICHClass	0.8911		**0.000905**	↗	**0.001095**	↗
Size and Polymorphism	CIS	0.4914		0.05132		0.1605	
	DAM	0.6724		**0.03264**	↗	**0.003362**	↗
	DSC	0.5031		0.4196		0.8725	
	EIC	0.6013		0.4277		0.5616	
	EIP	0.1874		0.5998		0.1039	
	MFA	0.9776		0.2374		0.243	
	MOA	0.9682		**0.01269**	↗	**0.01493**	↗
	NAD	0.921		**0.00277**	↗	**0.003884**	↗
	NADExtended	0.8652		**0.008383**	↗	**0.005466**	↗
	NMD	0.4008		**0.01341**	↗	**0.0467**	↗
	NCP	0.7092		0.4407		0.1198	
	NMA	0.3501		0.1012		0.3157	
	NMDExtended	0.5107		**0.02384**	↗	0.05112	
	NMI	0.4371		**0.02397**	↗	0.2016	
	NMO	0.8188		**0.0006559**	↗	**0.0005408**	↗
	NOM	0.4008		**0.01341**	↗	**0.0467**	↗
	NOParam	0.8372		0.1123		0.1551	
	NOPM	0.2496		0.9574		0.2793	
	PIIR	0.588		0.7276		0.2846	
	PP	0.8226		0.1112		0.1468	
	REIP	0.87		0.4993		0.3336	
	RPII	0.4809		0.652		0.8614	
Complexity	McCabe	0.8881		**0.001085**	↗	**0.00063**	↗
	SIX	0.6163		**0.002085**	↗	**0.0008183**	↗
	WMC1	0.4008		**0.01341**	↗	**0.0467**	↗
	WMC	0.9252		**0.0003315**	↗	**0.001297**	↗
Changeability and Rank	Class Rank	0.2598		0.9978		0.212	
	Frequencies of Changes in Past	0.9956		0.08194		0.08794	
	Frequencies of Changes in Future	0.9733		0.5469		0.5983	
	Numbers of Changes Past	0.212		**0.03508**	↗	0.06668	
	Numbers of Changes Future	0.8688		0.8537		0.7018	
Issues	Numbers of Issues	0.0728		0.1603		0.6645	

Table 7. p-values and Metrics Trends, with 0^{FP}. (A ↗ or ↘ represents an increase (respectively, decrease) of, for example in the third column, the metrics values of 1-role classes *wrt.* to these of 0-role classes.)

Metric Groups	Metric Names	1 role vs. 0 role		2 role vs. 0 role		2 role vs. 1 role	
		p-values	Trends	p-values	Trends	p-values	Trends
Changeability	Frequencies of Past Changes	**8.26E-07**	↗	**1.24E-09**	↗	0.08794	
	Frequencies of Future Changes	**0.0001564**	↗	**7.44E-06**	↗	0.5983	
	Numbers of Past Changes	**3.54E-07**	↗	**5.50E-10**	↗	0.06668	
	Numbers of Future Changes	**0.001552**	↗	**9.72E-05**	↗	0.7018	
Cohesion	CAM	0.854		**0.0001996**	↗	**0.0003884**	↗
	cohesionAttributes	0.6881		**0.04051**	↗	**0.0009488**	↗
	LCOM1	**0.01313**	↘	**6.22E-09**	↗	**0.0009946**	↗
	LCOM2	**0.01087**	↘	**1.41E-07**	↗	**0.0017**	↗
	LCOM5	**0.03454**	↗	**3.95E-06**	↗	**0.001383**	↗
Complexity	McCabe	0.2274		**7.85E-07**	↗	**0.00063**	↗
	SIX	**0.004657**	↗	**1.41E-08**	↗	**0.0008183**	↗
	WMC1	**2.09E-05**	↗	**4.00E-08**	↗	**0.0467**	↗
	WMC	**0.01453**	↘	**5.40E-07**	↗	**0.001297**	↗
Coupling	ACAIC	0.1733		**0.03935**	↗	0.5029	
	ACMIC	0.284		**0.002702**	↗	**0.04961**	↗
	CBO	0.5706		**0.0001434**	↗	**0.001948**	↗
	CBOin	0.191		**7.89E-06**	↗	**0.0005939**	↗
	CBOout	0.1055		**5.96E-07**	↗	**0.0001025**	↗
	connectivity	0.5005		0.07963		0.2603	
	CP	0.9802		0.2272		0.1428	
	DCAEC	**9.37E-06**	↗	**0.003612**	↗	0.06724	
	DCC	0.4149		**2.98E-05**	↗	**0.002347**	↗
	DCMEC	**0.0001468**	↗	**0.001024**	↗	0.595	
	PP	0.829		0.1382		0.1468	
	RFP	**0.04845**	↗	**0.01477**	↗	0.6074	
	RRFP	0.0968		**0.02306**	↘	0.5106	
	RRTP	**0.02637**	↘	**0.03722**	↘	0.6952	
	RTP	0.2005		**0.01295**	↗	0.3693	
Inheritence	AID	0.126		**0.0001542**	↗	0.1391	
	ANA	0.3958		0.8077		0.3918	
	CLD	**< 2.2e-16**	↗	**7.94E-11**	↗	**0.003298**	↘
	DIT	0.08713		**8.59E-05**	↗	0.2632	
	NCM	**0.00087**	↗	**4.84E-09**	↗	0.07486	
	NOC	**2.22E-16**	↗	**3.55E-11**	↗	0.245	
	NOD	**2.22E-16**	↗	**5.29E-11**	↗	0.07351	
	NOH	0.5644		0.601		0.9663	
	NOP	0.2248		**6.10E-06**	↗	**0.007146**	↗
	ICHClass	**0.03035**	↗	**2.03E-07**	↗	**0.001095**	↗
Issues	Numbers of Issues	**0.0003619**	↗	**0.0003612**	↗	0.6645	
Polymorphism and Size	CIS	**9.22E-07**	↗	**1.50E-08**	↗	0.1605	
	DAM	0.1285		**1.94E-05**	↗	**0.003362**	↗
	DSC	0.1461		0.2098		0.8725	
	EIC	**0.0002848**	↗	**9.03E-06**	↗	0.5616	
	EIP	**7.26E-13**	↗	**1.43E-09**	↗	0.1039	
	MFA	0.1138		0.7105		0.243	
	MOA	**0.0001883**	↗	**6.44E-10**	↗	**0.01493**	↗
	NAD	0.1349		**5.03E-06**	↗	**0.003884**	↗
	NADExtended	0.1514		**1.14E-05**	↗	**0.005466**	↗
	NCP	**5.39E-06**	↗	**0.01465**	↗	0.1198	
	NMA	**9.34E-06**	↗	**2.30E-06**	↗	0.3157	
	NMD	**2.09E-05**	↗	**4.00E-08**	↗	**0.0467**	↗
	NMDExtended	**3.37E-05**	↗	**1.07E-07**	↗	0.05112	
	NMI	0.1029		**0.0001075**	↗	0.2016	
	NMO	**0.00163**	↗	**3.57E-10**	↗	**0.0005408**	↗
	NOA	0.1868		**7.35E-08**	↗	**0.01153**	↗
	NOM	**2.09E-05**	↗	**4.00E-08**	↗	**0.0467**	↗
	NOParam	**7.81E-06**	↗	**2.38E-08**	↗	0.1551	
	NOPM	**2.89E-14**	↗	**1.93E-10**	↗	0.2793	
	PIIR	**7.00E-05**	↗	**0.01216**	↗	0.2846	
	REIP	**5.94E-10**	↗	**7.54E-08**	↗	0.3336	
	RPII	0.1486		0.08605		0.8614	
Ranking	Class Rank	**7.33E-09**	↗	**4.08E-06**	↗	0.212	

Table 8. p-values and Metrics Trends. (A ↗ or ↘ represents an increase (respectively, decrease) of, for example in the third column, the metrics values of 1-role classes compared to these of 0-role classes.)