






An Empirical Study on the Relationship Between the Co-Occurrence of Design Smell and Refactoring Activities

Lerina Aversano¹^a, Mario Luca Bernardi¹^b, Marta Cimitile²^c,
Martina Iammarino¹^d and Debora Montano³^e

¹University of Sannio, Department of Engineering, Benevento, Italy

²Unitelma Sapienza University, Rome, Italy

³Universitas Mercatorum, Rome, Italy

Keywords: Software Evolution, Software Quality, Refactoring, Design Smells, Statistical Analysis, Java Software.

Abstract: Due to the continuous evolution of software systems, their architecture is subject to damage and the formation of numerous design issues. This empirical study focuses on the co-occurrence of design smells in software systems and refactoring activities. To this end, a detailed analysis is carried out of the data relating to the presence of Design Smells, the use of refactoring, and the consequences of such use. Specifically, the evolution of 17 different types of design odors in five open-source Java software projects has been examined. Overall, the results indicate that the application of refactoring is not used by developers on design smells. This work also offers new and interesting insights for future research methods in this field.

1 INTRODUCTION


Delivering software changes within a very short time frame exposes their architecture to deterioration and often leads to design flaws, which is a hotly debated topic in software engineering. Several research papers address design challenges and their influence on the evolution of the software system. These studies underline how these problems increase the development costs of software systems over time until the software system itself becomes economically unsustainable.


Design smells are intrinsically linked to the evolution of the system as they relate to difficulties encountered during the design phase or the introduction of new features within the system. Furthermore, design smells are often produced as a result of inefficient design and architecture decisions that affect internal system attributes such as maintainability, extensibility, testability, understandability, and reusability (Le and Medvidovic, 2016). Failure to manage design smells could result in technical issues that can also


increase maintenance efforts and not just costs.


This empirical study examines the components of the source code in over 7,000 commits belonging to the evolutionary history of five open-source software systems. Specifically, design smells are identified in the source code of systems as they evolve and the related refactoring actions. The objective of the investigation is to examine the potential influence of refactoring on source code, focusing on 17 different types of design smells classified into four primary categories identified in the literature: abstraction, encapsulation, modularization, and hierarchy, (Fowler, 2002). The results show that elements of the source code affected by design smells are more likely to change, but developers rarely make these changes through refactoring operations. This implies that the absence of design smell is not related to the presence of refactoring. Furthermore, the contribution of our work is the new and innovative approach in the field of software, represented by the use of factor analysis and in particular by Multiple Factor Analysis (MFA) to support the results.


The paper is structured as follows: Section 2 reports the research related to our study, Section 3 explains the data extraction process, the features model, and the objectives. The results are reported in Section

^a <https://orcid.org/0000-0003-2436-6835>

^b <https://orcid.org/0000-0002-3223-7032>

^c <https://orcid.org/0000-0003-2403-8313>

^d <https://orcid.org/0000-0001-8025-733X>

^e <https://orcid.org/0000-0002-5598-0822>

4, while Section 5 lists the threats that could affect them. Finally, Section 6 reports the most important conclusions and suggestions for future work.

2 RELATED WORK

According to Fowler, refactoring is a discipline used to redesign an existing body of code, updating its internal structure without altering its external (Fowler, 2018) behavior. Hence, the goal of this strategy is to improve defects in the source code by modifying only the lines of code and not the final product. A collection of hundreds of code improvement ideas was also created, the goal of which is to help programmers use refactoring to eliminate code smells. In this regard, the impact of refactoring on the elimination of code smells and the changes in quality metrics following its adoption has been the subject of increasing research in recent years (Cedrim et al., 2016; Chavez et al., 2017; Mkaouer et al., 2017). The majority of programmers, unfortunately, have a knowledge of smells is purely theoretical. This is also because most defects do not occur as specified by the theory, so developers can only detect them when the smell occurs exactly as stated in its description. Fernandes et al. found that refactoring is only used for source code with at least one critical (Fernandes et al., 2020) attribute. Furthermore, it has been shown that the amount of smells increases if a change is introduced in the source code and if the refactoring is applied (Aversano et al., 2020b; Aversano et al., 2020a). Bibiano et al. show that batch refactoring techniques in 51% of the situations introduce new smells and in 38% of the occurrences it does not remove smells from the code (Bibiano et al., 2019). Further research investigated the usefulness of refactoring techniques to reduce the severity of design smells, although even in these circumstances this strategy proved ineffective (Saika et al., 2016; Aversano et al., 2007). According to much research, refactoring is more beneficial for improving software quality metrics than for removing smells from source code. Among all forms of smells, design smells in particular are typically indicative of architectural flaws in the code, and they are frequently a sign of damaging software maintainability (Garcia et al., 2009).

At this point in the literature, it is crucial to understand the power of refactoring on design smells. So most empirical studies show that refactoring, in general, cannot remove smells (Sharma et al., 2020; Aversano et al., 2022).

The research proposed in this paper aims to further investigate the relationships among refactoring

actions and design smells at a fine grained level by using factor analysis, statistical indices, and tests, a fresh and cutting-edge method in the software research.

3 APPROACH

This section outlines the main phases of the conducted empirical study, that are:

- definition of the study's research questions;
- data extraction and data collection;
- identification of the features to analyze.

Each phase is better explained in the following subsections.

3.1 Research Questions

This study aims to investigate the co-occurrence between design smells and refactoring activities, to understand whether smelly code is more prone to changes due to refactoring.

Overall, the following research questions have been identified:

RQ₁: *Do programmers refactor when Design Smells occur?*

This first research question aims to understand if there is a correlation between the instances of design smells with the use of refactoring by the developers, studying their presence in the commit.

RQ₂: *To what extent the use of refactoring change as the types of design smells to be managed increases?*

The second research question aims to better highlight the co-occurrence of the two phenomena, we have investigated whether the use of refactoring activities is greater where there are more design smells to manage.

RQ₃: *Is there any correlation between the category of design smells and refactoring?*

Different categories of design smells have been considered in the third research question to understand how they are specifically related to refactoring.

3.2 Data Extraction

The data used in this research comes from five open-source Java software projects. Table 1 shows the names of the projects evaluated in the first column, in the second column the number of commits analyzed,

Table 1: Software Projects characteristics.

System	#commit	First and Last Commit Date
Atlas	1581	08/12/2014 - 05/07/2019
Guice	1176	16/11/2006 - 06/06/2019
JUnit4	1406	03/12/2000 - 20/06/2019
Log4j	2042	14/12/2000 - 11/02/2014
Zookeeper	1084	06/11/2007 - 17/07/2019
Total	7289	03/12/2000 - 17/07/2019

and in the last column the time interval to which they belong. These projects have been chosen because their repositories are publicly available on GitHub¹, vary in size, and cover different domains.

The data collection process involved several stages. During the first phase, the elements of the source code to be analyzed have been extracted from the commits of each system. Specifically, the extracted source code has been given as input to the tools to detect the existence of smells and the presence of refactoring. Finally, each commit has been examined by comparing it to the previous one to see if the smell had been added or removed. More precisely, two different tools have been adopted: Designite² and RefactoringMiner³. The former was used to detect smells and is able to assess the quality of the design and thus recognize the existence of design smells. Consequently, for the sake of completeness, in Table 2 we provide the design smells detected by this tool for this study, identifying the list of smells in the first column and the macro-categories of design smells in the second. The second tool detects the existence of refactoring actions performed on the source code. This⁴ is particularly competent for recognizing 40 different types of refactoring for actions on packages, classes, methods, features, arguments, and attributes. Finally, the complete data-set includes the history of all software systems described in Table 1, which consists of 7289 commits, 43009 different modifications of the source code, and 19 features related to each type of design smell, the occurrence of refactoring on the commit, and the specific class (the java file) and the date of the commit.

3.3 Features Description

The following features have been analyzed to understand the relationships between refactoring and design smells.

- **Refactoring:** a dichotomous feature that considers the value "true" when refactoring techniques

¹<https://github.com>

²<https://www.designite-tools.com>

³<https://github.com/tsantalis/refactoringMiner>

⁴At the time of writing, in version 2.1

Table 2: Type & Category of Design Smells Considered.

Type of smells	Design smell Category
Imperative Abstraction	ABSTRACTION
Multifaceted Abstraction	
Unnecessary Abstraction	
Unutilized Abstraction	
Deficient Encapsulation	ENCAPSULATION
Unexploited Encapsulation	
Broken Modularization	MODULARIZATION
Cyclically Dependent Modularization	
Insufficient Modularization	
Hub-Like Modularization	
Broke Hierarchy	HIERARCHY
Cyclic Hierarchy	
Deep Hierarchy	
Missing Hierarchy	
Multipath Hierarchy	
Rebellious Hierarchy	
Wide Hierarchy	

are employed for its reference commit, and the value "false" when they are not.

- **Design Smells:** 17 different types of design smells have been detected. Each of these 17 characteristics is named after the design smell it represents. All the design smells considered in this paper are listed in Table 2. When a certain type of smell is present in the commit, it is represented with value '1', otherwise with value '0'. Hence, they are dichotomous characteristics.
- **Date:** the date on which the current commit was made.

3.4 Data Analysis

In this study, different statistical techniques have been used to analyze the various aspects of the research questions. In particular, the following methods have been used:

- *Row profiles* are widely used in statistical analysis when we are interested in understanding whether there is some conditioning relationship between two features, that is when one of the certain character conditions occurs, the other assumes a certain behavior. There are two measures for calculating this relationship: row and column profiles. In our study, we use row profiles because the distribution is conditioned concerning a precise modality of the row variable. A row profile is obtained by relating the frequency of the conditional distribution to the relative total (marginal frequency):
- *Spearman Rho non-parametric* is a rank-based correlation coefficient. To evaluate whether the determined coefficient is significant concerning the fixed alpha values, the obtained value is compared with the critical values of the Rho Spear-

man table. In our case study, we have formulated only one correlation hypothesis, which is non-independence therefore our test is unidirectional, for this reason, the rho value is significant if it exceeds the absolute value of the critical value reported in the table for $\alpha / 2 = 0.025$.

- *Multiple Factor Analysis* (MFA) is a multivariate technique used to summarize and illustrate complex data sets with naturally organized variables (quantitative and/or qualitative) (Escofier and Pagès, 1994; Le Dien and Pagès, 2003). It is a combination of a Principal Component Analysis (PCA) and a Multiple Correspondence Analysis (MCA) (Ringnér, 2008). During analysis, sets of variables are analyzed simultaneously, which requires balancing the effects of each set of variables. The number of variables in each group can be different. Consequently, during the survey, the variables of the same group are normalized using the same weighting value, which can vary from group to group. To be more precise, MFA attributes to each variable of the j -th group a weight equal to the inverse of the first eigenvalue of the analysis of the j group, an eigenvalue that is calculated with a PCA if the variable is quantitative or with MCA if the variable is qualitative. Through the eigenvalue, it is also possible to evaluate how much an observation, a variable or an entire table contributes to the inertia recovered by a component to better understand the interactions between components, observations, variables, and tables as well as help in the interpretation of a component.

4 EMPIRICAL STUDY FINDINGS

This section reports the experimental results obtained thanks to the use of the methodology and the model of features previously described.

4.1 Do Programmers Refactor When Design Smells Occur?

To examine how refactoring is used on smelly commits, we looked at the commits where design smells are found to see if there is a relationship to using refactoring where there are smells. In particular, the analyzes were conducted using the *Row Profiles* described in Section 3.4. The analyzes were performed on the total data-set, which contained instances of all the software systems chosen and on each one separately, to produce both an overview and a particular vision linked to a single software system.

Table 3: Row profiles of commits affected by design smells - ALL SOFTWARE.

TYPE OF DESIGN SMELL	REFACTORING ACTIVITY	
	No	Yes
Imperative Abstraction	90%	10%
Multifaced Abstraction	76%	24%
Unnecessary Abstraction	97%	3%
Unutilized Abstraction	79%	21%
Deficient Encapsulation	72%	28%
Unexploited Encapsulation	70%	30%
Broken Modularization	78%	22%
Cyclic Dependent Modularization	68%	32%
Insufficient Modularization	70%	30%
Hub like Modularization	61%	39%
Broken Hierarchy	75%	25%
Cyclic Hierarchy	86%	14%
Deep Hierarchy	100%	0%
Missing Hierarchy	70%	30%
Multipath Hierarchy	69%	31%
Rebellious Hierarchy	82%	18%
Wide Hierarchy	76%	24%

Table 3 shows the type of design smell in the first column, the row profiles that describe the percentage of commit in which refactoring was used (second column) and in which no was used (third column). Results in bold show that refactoring is not used in at least 60 % of commits affected by design smells. In particular, with the smell of Deep Hierarchy, refactoring is never used by developers while the type of smell on which refactoring is more used than the other types of design smell is Hub-like Modularization where in 40 % of commits affected, refactoring activities are used. We have also looked at the row profiles on the individual software systems to verify what we found in the overall analysis of all systems together. The results are reported in Table 4 in which the first column shows the types of design smells, from the second onwards the columns are organized in pairs, and each pair of columns shows the row profiles that characterize the proportion of commits where refactoring has been applied (the first column of the two) and where it has not been applied (the second column of the two).

Table 4 shows that also on the single system, in general, refactoring is not so practiced. In each software project, there are some design smells where the refactoring activities on dirty commits are higher than in the general analysis. This condition is normal because the single software system has its peculiar characteristics. In particular, the most refactored design smells for Atlas are Multifaced Abstraction (59%), Cyclic Hierarchy (64%), and broken Modularization (79%); for Guice are Insufficient Modularization (62%), Broken Hierarchy (69%), Deficient Encapsulation (84%) and Broken Modularization (92%); and for Log4j are Missing Hierarchy (56%) and Wide Hierarchy (78%). JUnit4 and Zookeeper are the only two software systems where the refactoring activities

are less used concerning the others systems.

All the analyzes conducted in this Section have been statistically examined, in particular, we used the non-parametric Spearman Rho, and all the results have a *p-value* minor to 0.025 so we can say that the analyzes demonstrate independence between the rows profiles and their statistical significance.

SUMMARY: Refactoring is less practiced on commits affected by design smells, additionally, it is never used in the case of rare smells like Deep Hierarchy.

4.2 To What Extent the Use of Refactoring Change as the Types of Design Smells to Be Managed Increases?

To understand how the use of refactoring changes based on the number of types of smells present in the class, we report the results of the relative analysis in Table 5 which presents in the first column the number of different types of design smells present into the class, in the second column the percentage of different types of design smells present in the classes not refactored, and in the third column the percentage for the classes where refactoring activities are used. In the analyzed classes we have found that there are at most 4 different types of smells, in particular, 59% of them have only one type of design smells and in 29% of the cases 2 types of different smells. Table 5 also highlights that 75% of the time refactoring is not applied regardless of the number of different types of design smells. Only 2% of the time refactoring is used to manage design smells and, in particular, it is mostly used to manage classes with 1(14%) or 2(8%) types of different design smells.

The same analyzes conducted on the individual software systems show the same empirical evidence there are often at most 2 types of different design smells in each class, and the cases in which more than 2 are present are very sporadic. Also, more than half the time, regardless of how many design smells come up, programmers don't resort to refactoring. The Atlas software system represents the only case in which the presence or absence of refactoring activities has substantially the same behavior (47% vs 53%). Furthermore, in this system, it is preferred to use refactoring when there are too many designs smells to manage, in fact when 4 different types of smells occur refactoring is used 8% of the time, instead only 1% of the time when 4 different types of design smell ap-

pear developers prefer not to use refactoring. These results are shown graphically in Figure 1 which shows on the abscissa axis the number of different types of design smells present in the classes and their percentage distribution on the ordinate axis. Therefore, the blue bars represent the classes where refactoring is practiced, while the orange bars represent the classes where refactoring is recognized. The graph for the total data-set is on the top left, followed by those for Atlas, Guice, JUnit4, Log4j, and finally Zookeeper. The proposed analyzes were statistically validated through the non-parametric Spearman Rho, a rank-based correlation coefficient, and with a correlation hypothesis, which is non-independence. Consequently, having always obtained a *p-value* minor to 0.025, we can conclude that the analyzes illustrate their independence and statistical significance.

SUMMARY: In most instances, refactoring is chosen when the source code contains few design smells, in particular, at most 2.

4.3 Is There Any Correlation Between Design Smells Categories and Refactoring?

To understand the correlation between different smell categories and refactoring activity, the possible presence of a common structure in the design smell categories, and the specificity of each, we have used Multiple Factor Analysis (MFA). The data-set contains 17 different characteristics for design smells, divided into 4 different smell groups: Abstraction, Encapsulation, Modularization, and Hierarchy. Therefore, MFA has been performed on the four categories of design smells and the presence of refactoring.

The graphs in Figure 2 illustrate the coordinates of the groups which provide information on the correlation between the groups. At the top left, we report the overall graph of the data set, while the graphs for Atlas, Guice, JUnit4, Log4j, and Zookeeper are shown respectively below. The coordinates of each group in the Cartesian axis give information on the correlation between the groups. In particular, if the coordinates of the groups are arranged along the same axis, the correlation between these groups is greater than that with the other scattered groups in the graph. The results show that the coordinates of the points that take into account the refactoring are the Modularization and the Abstraction which are the closest so they show that the refactoring activities are more correlated and connected with the presence of design smells belonging to the Modularization and Abstraction.

Table 4: Row profiles of commits affected by design smells for SINGLE SOFTWARE.

TYPE OF DESIGN SMELL	ATLAS		GUIDE		JUnit4		LOG4J		ZOOKEEPER	
	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY	REFACTORING ACTIVITY
	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Imperative Abstraction	58%	42%	0%	0%	0%	0%	82%	18%	85%	15%
Multifaced Abstraction	41%	59%	55%	45%	85%	15%	82%	18%	100%	0%
Unnecessary Abstraction	100%	0%	100%	0%	98%	2%	76%	24%	100%	0%
Untutilized Abstraction	63%	37%	41%	59%	76%	24%	79%	21%	83%	17%
Deficient Encapsulation	54%	46%	16%	84%	76%	24%	69%	31%	70%	30%
Unexploited Encapsulation	63%	37%	49%	51%	65%	35%	100%	0%	60%	40%
Broken Modularization	21%	79%	8%	92%	0%	0%	79%	21%	100%	0%
Cyclic Dependent Modularization	51%	49%	60%	40%	70%	30%	62%	38%	66%	34%
Insufficient Modularization	46%	54%	38%	62%	70%	30%	68%	32%	68%	32%
Hub like Modularization	48%	52%	97%	3%	0%	0%	0%	0%	0%	0%
Broken Hierarchy	64%	36%	31%	69%	68%	32%	63%	37%	75%	25%
Cyclic Hierarchy	36%	64%	95%	5%	100%	0%	0%	0%	0%	0%
Deep Hierarchy	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Missing Hierarchy	58%	42%	0%	0%	86%	14%	44%	56%	66%	34%
Multipath Hierarchy	54%	46%	93%	7%	0%	0%	0%	0%	0%	0%
Rebellious Hierarchy	100%	0%	0%	0%	89%	11%	54%	46%	69%	31%
Wide Hierarchy	93%	7%	0%	0%	85%	0%	22%	78%	65%	35%

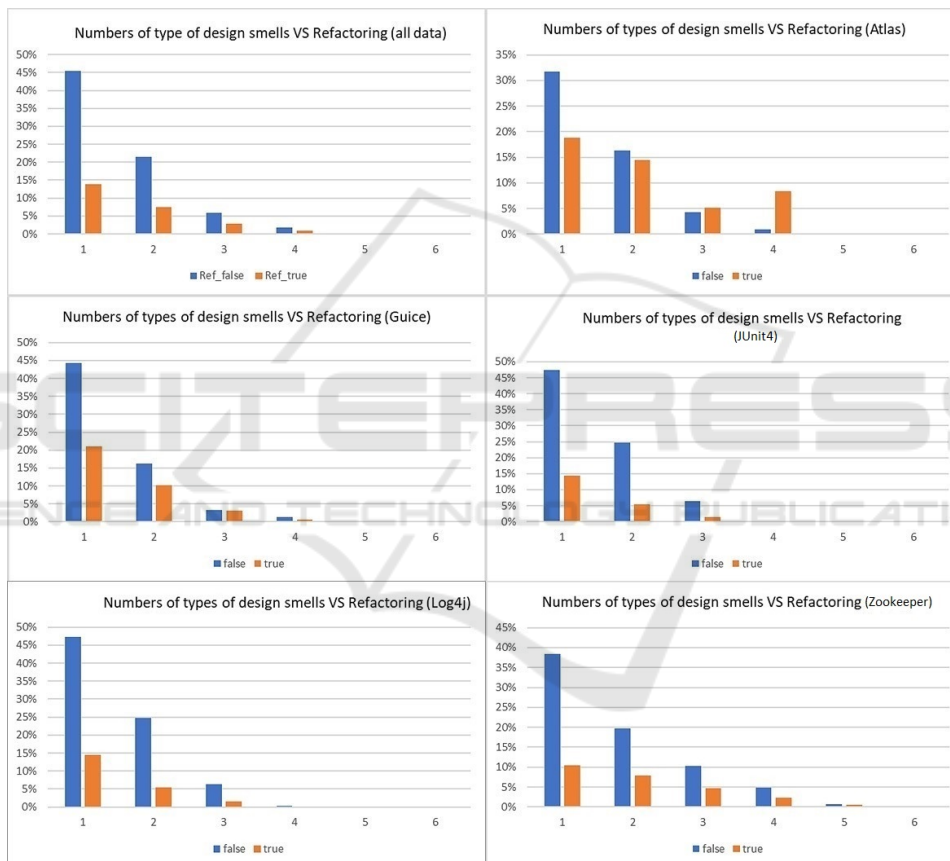


Figure 1: Distribution of Design Smells in the classes and Refactoring Activities - ALL DATA & SINGLE SYSTEM.

Table 5: Number of smells in the classes and refactoring activities - ALL SYSTEM.

#smell in a class	REFACTORING ACTIVITY		Total
	No	Yes	
1	45%	14%	59%
2	21%	8%	29%
3	6%	3%	9%
4	2%	1%	3%
5 or more	0%	0%	0%

SUMMARY: The categories of smells related to Modularization and Abstraction are those whose design smells are most closely associated with refactoring operations.

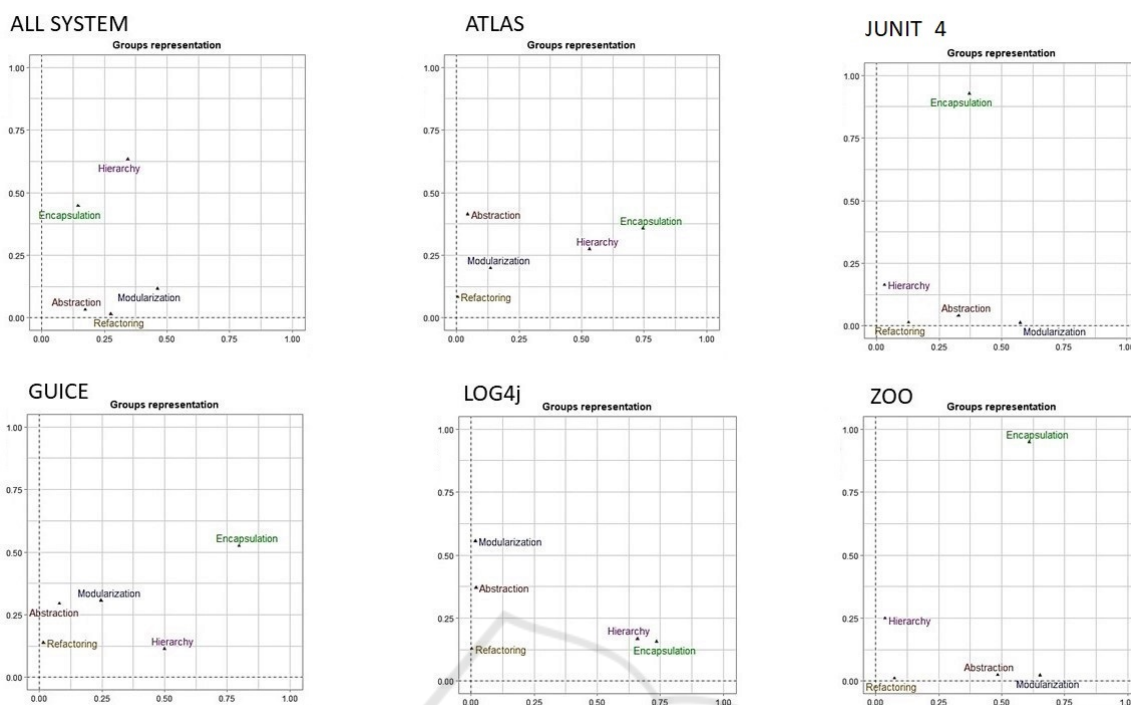


Figure 2: Multiple Factor Analysis (MFA) - ALL DATA & SINGLE SYSTEM.

5 THREATS TO VALIDITY

This section discusses the threats to the validity of the study described.

Threats to validity construction concern the relationship between theory and observation, so we can consider the potential inaccuracy of smell detection systems and refactoring activities. To reduce the threat, we used Designite, which has accuracy 96% and recall 99% (Tushar Sharma and Spinellis, 2020), and Refactoring Miner which has high accuracy 98% and recall 87% (Tsantalís et al., 2018).

Threats to internal validity relate to aspects of our study that could affect the results. We cannot assume that the cause-and-effect link between Design Smells refactoring and removal activities are always the same as determined by our research in general. To mitigate this vulnerability, we evaluated refactoring on a large number of commits (7289) of five pieces of software rather than just one. The potential lack of generalizability of the data produced poses a danger to the validity of the conclusions. To reduce this threat, we have considered five different systems. We performed analyzes on each system separately, taking into account its unique characteristics, as well as the combination of these, to obtain results for both the specific subset and the complete enumeration. This point is

also an external threat, in fact, we only looked at five software systems because this is the first phase of our research. The results can be confirmed or refuted during future research phases, taking into account multiple commits and multiple software systems.

6 CONCLUSION AND FUTURE WORK

This study suggests a different method for examining how design smells and refactoring operations interact. We have produced useful results for the scientific community using simple analyzes such as statistical techniques and factor analysis. The analyzes show that refactoring techniques are still underused in the field of software development. Furthermore, tests on single system software show that the frequency of using refactoring in smelly commits is significantly related to the nature of the design smell.

Refactoring is most frequently associated with the existence of design smells from the categories of Modularization and Abstraction. This is an interesting consideration that deserves to be explored across multiple systems in targeted research.

Future research will also focus on predicting analysis that can anticipate refactoring behavior in terms

of adding or eliminating design smells. To confirm the methodologies and results obtained in this study, it would be interesting to investigate how the use of refactoring varies according to the different programming languages of software projects.

REFERENCES

- Aversano, L., Bernardi, M., Cimitile, M., Iammarino, M., and Montano, D. (2022). Is there any correlation between refactoring and design smell occurrence? In *Proceedings of the 17th International Conference on Software Technologies - ICSOFT*, pages 129–136. INSTICC, SciTePress.
- Aversano, L., Bernardi, M. L., Cimitile, M., Iammarino, M., and Romanyuk, K. (2020a). Investigating on the relationships between design smells removals and refactorings. page 212 – 219.
- Aversano, L., Carpenito, U., and Iammarino, M. (2020b). An empirical study on the evolution of design smells. *Information (Switzerland)*, 11(11). Cited by: 4; All Open Access, Gold Open Access, Green Open Access.
- Aversano, L., Cerulo, L., and Del Grosso, C. (2007). Learning from bug-introducing changes to prevent fault prone code. page 19 – 26.
- Bibiano, A. C., Fernandes, E., Oliveira, D., Garcia, A., Kalinowski, M., Fonseca, B., Oliveira, R., Oliveira, A., and Cedrim, D. (2019). A quantitative study on characteristics and effect of batch refactoring on code smells. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE.
- Cedrim, D., Sousa, L., Garcia, A., and Gheyi, R. (2016). Does refactoring improve software structural quality? a longitudinal study of 25 projects. In *Proceedings of the 30th Brazilian Symposium on Software Engineering*, pages 73–82.
- Chavez, A., Ferreira, I., Fernandes, E., Cedrim, D., and Garcia, A. (2017). How does refactoring affect internal quality attributes? a multi-project study. In *Proceedings of the 31st Brazilian symposium on software engineering*, pages 74–83.
- Escofier, B. and Pagès, J. (1994). Multiple factor analysis (afmult package). *Computational Statistics & Data Analysis*, 18(1):121–140.
- Fernandes, E., Chávez, A., Garcia, A., Ferreira, I., Cedrim, D., Sousa, L., and Oizumi, W. (2020). Refactoring effect on internal quality attributes: What haven't they told you yet? *Information and Software Technology*, 126:106347.
- Fowler, M. (2002). Tutorials-refactoring: Improving the design of existing code. *Lecture Notes in Computer Science*, 2418:256–256.
- Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.
- Garcia, J., Popescu, D., Edwards, G., and Medvidovic, N. (2009). Toward a catalogue of architectural bad smells. In *International conference on the quality of software architectures*, pages 146–162. Springer.
- Le, D. and Medvidovic, N. (2016). Architectural-based speculative analysis to predict bugs in a software system. In *Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16*, page 807–810, New York, NY, USA. Association for Computing Machinery.
- Le Dien, S. and Pagès, J. (2003). Hierarchical multiple factor analysis: application to the comparison of sensory profiles. *Food Quality and Preference*, 14(5):397–403. The Sixth Sense - 6th Sensometrics Meeting.
- Mkaouer, M. W., Kessentini, M., Cinnéide, M. Ó., Hayashi, S., and Deb, K. (2017). A robust multi-objective approach to balance severity and importance of refactoring opportunities. *Empirical Software Engineering*, 22(2):894–927.
- Ringnér, M. (2008). What is principal component analysis? *Nature biotechnology*, 26(3):303–304.
- Saika, T., Choi, E., Yoshida, N., Haruna, S., and Inoue, K. (2016). Do developers focus on severe code smells? In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 4, pages 1–3. IEEE.
- Sharma, T., Singh, P., and Spinellis, D. (2020). An empirical investigation on the relationship between design and architecture smells. *Empirical Software Engineering*, 25(5):4020–4068.
- Tsantalís, N., Mansouri, M., Eshkevari, L., Mazinianian, D., and Dig, D. (2018). Accurate and efficient refactoring detection in commit history. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 483–494. IEEE.
- Tushar Sharma, P. S. and Spinellis, D. (2020). An empirical investigation on the relationship between design and architecture smells. *Empirical Software Engineering*, 25:4020–4068.