Received: 12 June 2023 | Revised: 17 December 2023 | Accepted: 23 February 2024

DOI: 10.1002/smr.2666

RESEARCH ARTICLE - EMPIRICAL

Journal of Software: Evolution and Process WILEY

An empirical investigation of the relationship between pattern grime and code smells

Maha Alharbi[1,2] | Mohammad Alshayeb[1,3]

[1]Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

[2]Department of Computer Science, University College of Al Jamoum, Umm Al Qura University, Mecca, Saudi Arabia

[3]Interdisciplinary Research Centre for Intelligent Secure Systems, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Correspondence
Mohammad Alshayeb, Interdisciplinary Research Centre for Intelligent Secure Systems, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
Email: alshayeb@kfupm.edu.sa

Funding information
No funding was received for conducting this study.

Abstract

Developers are encouraged to adopt good design practices to maintain good software quality during the system's evolution. However, some modifications and changes to the system could cause code smells and pattern grime, which might incur more maintenance effort. As the presence of both code smells and pattern grime is considered a bad sign and raises a flag at code segments that need more careful examination, a potential connection between them may exist. Therefore, the main objective of this paper is to (1) empirically investigate the potential relationship between the accumulation of pattern grime and the presence of code smells and (2) evaluate the significance of individual code smells when they appear in a specific pattern grime category. To achieve this goal, we performed an empirical study using six-grime metrics and 10 code smells on five Java open-source projects ranging from 217 to 563 classes. Our statistical results indicate that, in general, the growth of grime is more likely to co-occur with code smells using Spearman's correlation and Odd Ratio test. Specifically, there is a strong positive association between the growth of pattern grime at the class level and the presence of Shotgun Surgery smell according to the result of applying the Apriori algorithm, which gives conviction values equal to 1.66. The findings in this paper are helpful for developers and researchers as the presence of pattern grime could be considered a factor in improving the performance of existing smell detection methods. Furthermore, the link between grime and smells can be exploited as a hint for smell distribution in the system.

KEYWORDS
code smells, design patterns, pattern grime, software quality

1 | INTRODUCTION

Well-designed software pertained to good software quality and vice versa.[1] Modularity, reusability, and maintainability are some of the key factors that characterize a well-designed software or system. One of the most popular techniques developers resort to in order to achieve these characteristics is incorporating design patterns into the software design phase. Using design patterns in software design has been proven to improve the efficiency and quality of software development, especially if they are utilized in the early stages of the software development life cycle before reaching the implementation phase.[2] That is because considering good design practices at early stages of software development, such as the design phase, can reduce the risk of costly rework and effort to improve the overall maintainability of the software as compared with applying design patterns later in the development process during the implementation phase or software maintenance. Design patterns are

proposed solutions for recurring design problems. Gamma et al.[3] defined a catalog consisting of 23 design patterns that became a popular reference for software developers to adopt good design practices. Software developers should understand how design patterns work to employ them properly in the system under development. During the system's evolution, developers propose several changes for many reasons, such as responding to new functionality requests and maintenance issues. These new modifications may affect some components related to design patterns and add new elements and associations unrelated to the essence of the patterns and their responsibilities. In some cases, the addition of new elements is harmful as it increases the coupling between design pattern components and these new elements, which leads to the accumulation of artifacts that are not related to the definition of the design patterns. As the coupling increases in components participating in design patterns, it becomes harder for developers to make a change without affecting the related components, which reduces the efficiency of design patterns and incurs negative consequences on the pattern testability and maintainability. This case is known as pattern grime, which was defined by Izurieta and Bieman[4] as follows: "degradation of a design pattern instance due to accumulation of artifacts unrelated to the instance." Even though the main goal of using design patterns is to improve system maintainability, the presence of pattern grime in design pattern instances is harmful as it makes the pattern hard to maintain and change. Noteworthy, some modifications and changes to the system are unavoidable in order to meet system requirements. However, suppose the new changes imposed unrelated artifacts to the pattern responsibilities. In that case, this will be considered grime as it requires additional effort from developers to realize the distortion that occurred in the definition of the original pattern. Izurieta and Bieman[5] classified pattern grime into three main categories from a structural perspective: class grime, modular grime, and organizational grime. Class grime is the accumulation of class elements, such as methods and fields unrelated to the responsibilities of classes participating in design pattern instances. Modular grime is the accumulation of unnecessary relationships within the classes participating in design pattern instances. Organizational grime is the accumulation of grime as a result of unnecessary class distribution in packages, namespaces, or modules within a software system. This initial taxonomy was modified by extending new subtypes of class grime[6] and modular grime.[7] On the other hand, the term "pattern instances" in the context of software design refers to a code element that follows particular implementation practices of a design pattern. Identifying pattern instances involves recognizing structural and behavioral aspects in the code that match the defined characteristics of a design pattern. The elements that constitute the implementation of a design pattern are considered to be inside pattern instances. This typically includes classes, interfaces, methods, and attributes that are directly involved in shaping the pattern's structure and behavior. The relationships between these elements, such as inheritance, composition, and method invocation, are also within the pattern instance, as they facilitate the pattern's designated functionality. However, code elements that do not participate in the implementation or functionality of the design pattern are outside of the instance boundaries.

Code smells are symptoms of bad design choices that appear at the code level, which may incur more maintenance effort. Code smells are defined by Fowler[8] as bad signs in the source code that indicate the presence of potential design problems, which may hinder software maintainability if not resolved. The rationale of design patterns could be seen as the opposite of code smells as the former imposes good design practices in the system, whereas the latter represents signs of code flaws. Moreover, code smells are similar to pattern grime in that their presence in the system is undesirable. Several studies showed that not all code smells are harmful and indicate real problems; in some cases, their presence just points to places that need more careful examination.[9–11]

Many studies investigated the connection between design patterns, code smells, and pattern grime from different aspects. However, most of the reported studies in the literature focused on the software quality aspect, for instance, investigating the impact of design patterns and code smells on software quality in terms of quality attributes such as fault-proneness, change-proneness, and maintenance effort.[12–14] Moreover, numerous researchers shed light on the relationship between the presence of pattern grime and various software quality attributes.[4,15–17] Even though many studies discussed the impact of design patterns, code smells, and pattern grime on software quality attributes, as mentioned above, no study has yet explored the relationship between pattern grime and code smells, which opens the door for more research endeavors to bridge this gap. Furthermore, only a few studies suggested a potential relationship with the presence of code smells as pattern grime accumulates in the software[15]; however, this claim was not supported with empirical evidence to rely on. As the presence of both code smells and pattern grime raises a flag at code segments that need more careful examinations, a potential connection between them may exist. Therefore, understanding the association between the accumulation of pattern grime and the existence of code smells deserves more rigorous investigations from an empirical perspective to understand the consequences of their co-occurrence presence on software quality for developers and researchers. This is the main motivation for conducting this study, as investigating such a relationship may reveal new guidelines and recommendations that would help researchers and developers refine best practices and eventually lead to better software quality. Therefore, the main objective of this study is twofold: (1) to empirically investigate the potential relationship between the accumulation of pattern grime and the presence of code smells and (2) to evaluate the significance of individual code smell when it appears in a specific pattern grime category. According to the aforementioned objectives, this study has two contributions: (1) provides empirical evidence based on statistical test analysis on correlating the accumulation of pattern grime and code smells and clarifies if the accumulation of pattern grime influences the presence of code smells and (2) empirically evaluates the smell proneness of each pattern grime category toward the investigated code smells and identifies which (pattern grime category, code smell) pair shows significant relationships, if any, and which pattern grime category has a strong impact on the presence or absence of code smells.

The outcome of this empirical study can raise the awareness of researchers and practitioners about the nature of the relationship between the pattern grime and code smells, which can guide them to propose new recommended practices or refine the existing ones. Moreover, it could reveal new insights that lead the researchers to improve the effectiveness of existing tools for code smells and pattern grime detection.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 demonstrates the study design and research methodology followed to conduct this empirical study. The main findings are demonstrated and discussed in Section 4. The implication of this work is discussed in Section 5. The threats to the validity of this study are presented in Section 6. Finally, Section 7 provides the conclusion and the direction for future research.

2 | RELATED WORK

This section provides an overview of empirical studies on code smells and pattern grime separately. To the best of our knowledge, no empirical study in the literature has investigated a possible correlation between pattern grime and code smells.

2.1 | Empirical studies on pattern grime

Researchers investigated the impact of pattern grime on several software quality attributes, such as testability, adaptability, and understandability. Izurieta and Bieman[15] performed an experiment on one open-source system to investigate the impact of pattern grime accumulation while the system evolves on design patterns' testability. Three design patterns were considered in this study: Singleton, State, and Visitor patterns. Their findings demonstrated that the buildup of pattern grime with the aging of software systems leads to an increase in the testability of these design patterns as the number of required test cases increased significantly as a result. They also suggested that early identification and refactoring of code smells, as well as early removal of pattern grime, could make the system easier to maintain and enhance its testability. As a continuation of this effort, Izurieta and Bieman[4] evaluated the relationship between the growth of pattern grime and the testability and adaptability of design pattern instances. They analyzed the growth of pattern grime in three open-source systems considering all three categories of pattern grime: class, modular, and organizational grime. As a result of multiple case studies, they concluded that the modular grime accumulation increased during the evolution of design patterns in several versions of the same project. Based on the analysis of their results, the authors suggest that the primary contributor to this grime accumulation is the increase of inter-class coupling observed within the classes participating in the design pattern implementations, thus justifying the presence of modular grime. Another empirical study was conducted by Izurieta and Griffith[6] to address more software quality attributes. The study aimed to investigate the impact of class grime accumulation on the understandability of pattern instances to help extend class grime taxonomy. They found that an increase in class grime accumulation leads to a decrease in pattern understandability, quantified as a mean change of understandability for each affected class. Reimanis and Izurieta[18] recently refined and extended the grime taxonomy that was built based on a structural perspective and added a new grime classification by considering the behavioral perspective. Furthermore, they evaluated the relationship between structural pattern grime and behavioral pattern grime using several controlled experiments. They found a strong positive correlation between certain types of structural and behavioral grime. During the experiments, they manually injected code in design pattern instances to represent each type of grime. They also analyzed the impact of the newly added behavioral grime on design pattern quality, and the findings showed an observed correlation between behavioral grime and the size of the design pattern.

Despite the significant progress in design pattern detection with the support of automatic tools, a few works have been done on developing an automated tool for pattern grime detection. Griffith[19] conducted several controlled experiments to empirically investigate and further expand the knowledge about the consequences of the existence of pattern grime in code. The main aim was not only to study the impact of pattern grime accumulation during the evolution of the system on its maintainability and adaptability but also to assess the relationship of each type of pattern grime with technical debts. To achieve this goal, Griffith proposed an approach for developing a semi-automated tool for detecting several types of pattern grime. Griffith proposed 26 subtypes of pattern grime, where each grime category is subdivided into further subtypes. Particularly, the class grime is subdivided into eight subtypes, the modular grime has six subtypes, and the organizational grime has 12 subtypes. However, this research is ongoing, and the tool has not been completely developed yet, and the final outcome of the study is a semi-automated tool rather than a fully automated tool. Feitosa et al.[17] addressed the correlation between pattern grime and three quality attributes, namely, performance, security, and correctness. To achieve this goal, a case study was performed on five industrial projects. They proposed a tool called Spoon-pttgrime* to detect the presence of structural pattern grime automatically, which is the first automated tool designed mainly for pattern grime detection. The result of this study showed a negative correlation between the accumulation of pattern grime and the investigated quality attributes. Abdelaziz et al.[20] proposed the Design Pattern Violations Identification and Assessment (DPVIA)† tool. The main goal of the DPVIA tool is to detect any violations that occurred to the existing design pattern realization and assess the detected violations against pre-defined characteristics to compute the conformance score. In addition, DPVIA can provide details of the violations supported with graphs and related refactoring recommendations to mitigate the impact of the detected violations. Table 1 compares this study against the most relevant studies mentioned in this section by

TABLE 1 A comparison of the current study with related research.

Reference	Study focus	Number of systems	Research methodology	Correlation
Izurieta and Bieman[15]	Impact of pattern grime on testability	1 open-source (JRefactory)	Case study	No
Izurieta and Bieman[4]	Growth of pattern grime vs. testability and adaptability	3 open-source (JRefactory, ArgoUML, and eXist)	Case study	No
Izurieta and Griffith[6]	Impact of class grime on design pattern understandability	1 open-source (Percerons component database)	Experiment (pilot study)	No
Reimanis and Izurieta[18]	Relationship between structural and behavioral grime and relationships between behavioral grime and pattern quality	5 open-source (Selenium, RxJava, guava, spring-boot, and Hystrix)	Case study	Yes
Griffith[19]	Consequences of pattern grime on maintainability and technical debts	14 open-source projects with several versions.	Experiment	No
Feitosa et al.[17]	Correlation between pattern grime and performance, security, and correctness	5 industrial projects	Case study	Yes
This Study	Investigate the relationship between pattern grime and code smells	5 open-source systems	Experiment	Yes

outlining the specific relationships each study examines, the number of systems used to evaluate the relationship, the adopted research methodology, and the usage of correlation analysis in their investigations.

2.2 | Empirical studies on code smells

Fowler initially identified code smells[8] as he described 22 code smells along with possible corrective ways for removing and refactoring them. Several studies after that contributed to enlarging this initial library of code smells. Kerievsky[21] extended Fowler's work and added five additional code smells. He also refined the definition of the original code smells and proposed different corrective refactoring. Mantyla[22] and Mantyla et al.[23] conducted an empirical study to investigate the relationship between the original 22 code smells and code quality evaluation. The study aimed to improve the understandability of the original taxonomy proposed by Fowler. Their work resulted in a more refined taxonomy that reclassified the original 22 code smells and mapped them to seven high-level categories based on their inter-relationships in a system. Alternative taxonomy was later proposed by Wake.[24] Furthermore, a Systematic Literature Review (SLR)[25] on the relationship between code smells and quality attributes identified 93 code smells from 74 primary studies. The result of this SLR revealed that seven out of 93 code smells were mostly explored in reported empirical studies, among the others which are Duplicate Code, God Class, Refused Bequest, Long Method, Long Parameter List, Feature Envy, and Data Class.

A large body of studies in the literature concerning the impact of code smells on software quality, and several empirical studies showed evidence that point to code smells being harmful to software maintainability. Olbrich et al.[26] examined the influence of God class and Brain class code smells on the quality of three large open-source systems. Their findings indicate that classes associated with the God class and Brain class code smells underwent more frequent modifications and contained more defects than other non-smelly classes, which required more maintenance effort. Olbrich et al.[27] explored the presence of code smells during the evolution of a system and how they influence the pattern of changes in terms of frequency and size. Their empirical investigations focused on two specific code smells: God Class and Shotgun Surgery. The results showed that classes that contained these two code smells suffered from more change frequency and significant increases in size, which increased the likelihood that maintaining these classes required more effort. Chatzigeorgiou and Manakos[28] and Zazworka et al.[29] extended this study by adding more code smells to validate the reported results and provide more generalizable evidence. Similarly, findings of other studies indicated that classes with code smells are more change-prone[13,30] and contain more defects[31–33] when compared with non-smelly classes.

After reviewing the literature, we can conclude that no empirical study has been conducted to investigate whether there is a relationship between pattern grime and code smells. Furthermore, the implicit association between code smells and pattern grime mentioned by Izurieta and Bieman[15] was considered without providing empirical evidence to support that claim. Hence, it would be interesting to empirically examine the relationship between the accumulation of pattern grime and the existence of code smells in the context of this study to draw generalizable evidence that may confirm or reject the claim mentioned above.

3 | EMPIRICAL STUDY DEFINITION AND PLANNING

The main objective of this study is *to empirically investigate the potential relationship between the accumulation of pattern grime and the presence of code smells*. The Goal-Question-Metric approach is followed to formulate the goal of this empirical study.[34] The GQM approach consists of three main levels. The first level is the conceptual level, which involves identifying the main goal of the experiment. The second level is the operational level, where questions are derived from the identified goal, and the answers to these questions are used to check if the goal is met or not. The last level is the quantitative level, which includes measures selected to answer each question. Following the above GQM template, the goal of this empirical study is defined as follows: Analyze the accumulation of pattern grime and code smells for the purpose of evaluating their co-occurrence with respect to its smell proneness from the point of view of the researcher in the context of five Java-based open-source projects.

In order to achieve the aforementioned objectives, three research questions have been addressed through this empirical study:

RQ1: What relationship, if any, exists between the accumulation of pattern grime and the presence of the investigated code smells?

The rationale of this research question is to provide empirical evidence on correlating the accumulation of pattern grime and code smells to clarify whether the accumulation of pattern grime may influence the presence of code smells or not. All three categories of pattern grime described in Section 1, which are class, modular, and organizational pattern grime, have been considered in this study to provide a comprehensive analysis. It is worth mentioning that the first research question investigates the potential existence and assesses the strength of the relationship between pattern grime and code smells with a bird's-eye view. However, a specific code smell may appear more frequently with a particular grime category, which led us to address the second research question.

RQ2: Are there particular code smells more frequently appear in a particular category of pattern grime?

Besides analyzing the nature of the relationship between the accumulation of pattern grime and code smells, addressed by RQ1, we are interested in whether a connection combines specific pattern grime category and individual code smell. This question evaluates the smell proneness of each pattern grime category toward the investigated code smells to identify which (pattern grime category, code smell) pairs show significant relationships, if any, and which pattern grime category strongly impacts the presence or absence of code smells. Investigating the significance of each code smell when it appears in a specific pattern grime category could reveal new insights that help the researcher improve the effectiveness of existing tools for code smells and pattern grime detection.

RQ3: What relationship, if any, exists between the accumulation of pattern grime and the presence of the different types of design patterns?

Even though the primary goal of this study is to investigate the relationship between pattern grime and code smells from different granularity, it is also interesting to know if certain types of pattern instances are more likely to correlate to the different types of grime. Based on the definition of design pattern, which reflects good design practices, and the definition of pattern grime, which indicates a degradation of design pattern instance, one may intuitively think that design pattern instances and grime instances are negatively correlated if the design pattern is correctly implemented and vice versa. Meaning that a well-implemented design pattern that closely follows the pattern's intent and structure is less likely to accumulate grime. However, an empirical investigation is required to answer this research question with empirical evidence on correlating different types of design pattern instances and grime pattern instances, which would clarify whether the accumulation of pattern grime may influence the presence of design patterns or not.

3.1 | Data collection and analysis methodology

Figure 1 summarizes the detailed steps followed in conducting this empirical study, which consists of six steps:

STEPS 1–2: To address the research questions and perform analysis, we first conducted a thorough literature search to identify projects that contain design patterns, as the existence of design patterns is a prerequisite to measuring the accumulation of pattern grime. The presence of the design pattern in the prospective projects should be validated for use in our study, ensuring their reliability for our research purposes. We also looked for variations in our study's sample in terms of the size of the project and the context in which each was applied. Having this kind of variation in the study sample reduces the bias of the results, facilitates the comparison process, and produces more robust empirical evidence. As a result of this search, we found the P-Mart benchmark repository,[35] which contains small to medium Java-based projects mainly developed for academic purposes and already implemented design patterns. Based on the methodology described in Figure 1, we have randomly selected five Java-based projects from the P-Mart repository that were also used in previous related empirical studies.[27,36,37] It should be noted that the

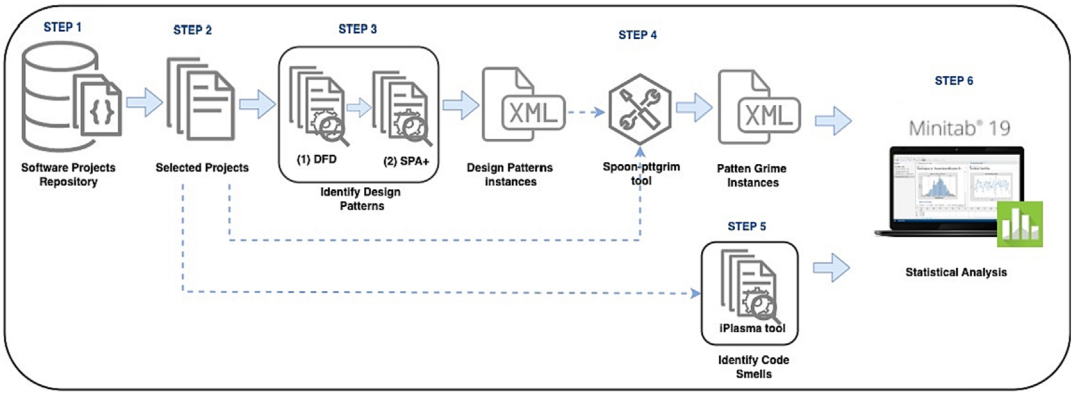

FIGURE 1 Detailed steps for research methodology followed in this study.

TABLE 2 Size of the selected five projects that were used in the experiments.

Projects	LOC	No. classes	Version no.
QuickUML[38]	18,400	217	v2001
MapperXML[39]	30,767	234	v1.9.7
Nutch[40]	32,421	283	v0.4
JhotDraw[41]	54,746	356	v7.0.6
Jrefactory[42]	93,113	563	v2.6.24
Total size (all projects)	136,334		

evolution of pattern instances in the selected projects is not considered in this study; we conducted the experiment on a single snapshot in time of each project. Table 2 lists the size of each project measured in terms of Line of Code (LOC) unit, number of classes, along with its version number. All projects used in this study are made available online[‡] for reproducibility purposes.

STEP 3: For each selected project, we need to collect pattern instances to establish a description list that serves as a baseline for distinguishing and measuring pattern grime in later phases. To make this process less error-prone, as our sample size is quite large, we searched the literature for existing tools that can automate the collection of pattern instances from the selected projects. Several tools have been developed and used in previous studies to detect design patterns in software projects; however, most have limitations that make them unsuitable for our specific research needs. For example, SPQR,[43] MARRPLE,[44] DP-Miner,[45] WOP,[46] and DPRE[47] are commonly used tools for detecting design patterns; however, they support a low number of design patterns that may not be able to detect all the design patterns present in our selected projects. Similarly, the low scores of recall or precision of MARRPLE,[44] WOP,[46] and DeMIMA[48] make them unreliable for our study. SPQR[43] and Pinot[49] did not mention the accuracy of their pattern detection capabilities. On the other hand, the Design Pattern Detector (DPD)[§] tool, developed by Tsantalis et al.,[50] can detect 11 different design patterns with high accuracy (recall: 95.9, precision: 100) as compared with the previously mentioned tools. Therefore, after careful consideration of the capabilities of existing tools, we decided to use the DPD tool for detecting design patterns in our selected projects. Moreover, SSA+[¶] tool, developed by Feitosa et al.,[17] was also considered in our study as it extends the capabilities of the DPD tool by differentiating instances of certain design patterns. Particularly, DPD can only detect abstract classes participating in design patterns. On the other hand, SSA+ complements this list by capturing the concrete classes participating in design patterns, which gives a more comprehensive realization of participating pattern instances in each project. The output of the DPD tool is used as an input for the SSA+, which finally generates five XML files, one for each project, that are used later for pattern grime assessment (i.e., to collect values of pattern grime metrics), meaning that the output of SSA+ is then used as an input in Step 4. Regarding the validation of SSA+ performance in terms of its accuracy, the authors mentioned that the result of the tool was manually validated by selecting 50 random instances and all of them were correctly identified patterns.[51] Table 3 summarizes the result of conducting Step 3 by listing design patterns identified in each project with their categories that belong to and the number of instances detected for each pattern.

STEP 4: To assess the presence of pattern grime in the selected projects, we followed the same approach as Feitosa et al.,[17] who proposed a tool called Spoon-pttgrime (v0.1.0) to quantify the amount of pattern grime automatically. Spoon-pttgrime accepts two files as input: (1) Java-based source code projects and (2) the XML file generated by SSA+ that represents the pattern instances in the selected projects in an Abstract Syntax

TABLE 3 List of design patterns detected in each project with the count of instances per each design pattern.

		# instances per project					
Design pattern category	**Design pattern type**	**QuickUML**	**MapperXML**	**Nutch**	**JhotDraw**	**Jrefactory**	**All projects**
Creational	Factory method	0	0	4	3	1	8
	Prototype	0	0	0	12	0	12
	Singleton	1	3	2	2	12	20
Structural	Adapter	6	11	18	26	26	87
	Bridge	0	0	0	13	1	14
	Composite	1	0	0	2	0	3
	Decorator	0	0	6	3	0	9
	Proxy	6	6	15	0	9	36
Behavioral	**Command**	**0**	**0**	**0**	**0**	**0**	**0**
	Observer	0	2	0	2	1	5
	State	5	15	14	51	16	101
	Strategy	**0**	**0**	**0**	**0**	**0**	0
	Template method	8	4	5	12	17	46
	Visitor	0	0	0	0	99	99
	Chain of responsibility	0	0	0	0	1	1

TABLE 4 Set of metrics used for pattern grime assessment along with their description and their grime category.[17]

Grime category	Metric	Description
Class	***cg-naa***	Number of alien attributes participating in pattern instance classes
	cg-napm	Number of alien public methods in pattern instance classes
Modular	***mg-ca***	Number of afferent couplings in pattern instance
	mg-ce	Number of efferent couplings in pattern instance
Organizational	***og-ca***	Number of afferent couplings in pattern instance (at the organizational level)
	og-np	Number of packages participating in pattern instance

Tree (AST) format. Therefore, our study samples were all written in Java language only. Spoon-pttgrime analyzes the source code of each project with respect to the corresponding produced XML file (generated from STEP 3) to compute the amount of grime based on six metrics, two for each of the three grime categories considered in this study: (1) class grime, (2) modular grime, and (3) organizational grime. Class grime is assessed by analyzing participant classes in pattern instances using (***cg-naa***) and (***cg-napm***), which compute the number of alien attributes and the number of alien public methods, respectively. On the other hand, modular grime is assessed by computing afferent coupling (***mg-ca***) and efferent coupling (***mg-ce***), which consider the number of dependencies or coupling relationships coming in or out of pattern instances. For organizational grime, afferent coupling (***og-ca***) is also used as a metric to assess the grime at the organizational level, which scans the presence of coupling relationships but with a higher view of abstraction. In addition to afferent coupling (***og-ca***), organizational grime can be depicted using another metric (***og-np***), which computes the number of packages participating in pattern instances. Table 4 lists the used metrics with their description and the category to which each one belongs.

STEP 5: After assessing the amount of pattern grime, we detected the presence of code smells to analyze their relationship, which embodies the essence of this study. For code smell detection, we used iPlasma.[#] There are several tools in the literature for code smell detection, such as JDeodorant,[52] JSpIRIT,[53] infusion,[**] and its extended version inCode.[54] However, we selected the iPlasma tool specifically over the other tools for several reasons. The first and most important reason is that iPlasma can detect almost 13 types of code smells, including seven types identified by Fowler, which provides a broader coverage of code smells compared with the other currently available tools. In addition, the iPlasma tool has been empirically validated and used in several related studies. Its performance and precision in code smell detection outperform the existing tools in the literature. Moreover, iPlasma is open-source and available online for public use, whereas the infusion is no longer available online, and its extended version, inCode, is only available for commercial use. Even though iPlasma can detect 13 types of code smells, in this study, we focused only on 10 types of code smells because these are the code smell types that have been found in the five selected projects. The 10 types of code

smells detected from the selected projects are God Class, Refused Parent Bequest, Intensive Coupling, Extensive Coupling, Feature Envy, Brain class, Brain Method, Shotgun Surgery, Futile Hierarchy, and Data Class. Seven out of the 10 detected code smells were mentioned in two SLRs as code smells were mostly explored in reported empirical studies, which keeps our study in alignment with most recent related studies.[25,55]

STEP 6: To address the first research question, we investigated the number of code smells detected in each pattern instance versus each grime metric computed for the same pattern instance in a pairwise manner (e.g., ***<No. code smells detected in pattern instance x, cg-naa in pattern instance x>***) to compute the correlation coefficient for each pair. The selection of the most appropriate correlation test was mainly based on evaluating the normality distribution assumption of the collected data. In this regard, two methods of correlation tests are widely used, which are Pearson's correlation coefficient method[56] and Spearman's rank correlation coefficient method,[56] where the former is used when data are normally distributed, and the latter is used when the assumption of normality is not satisfied. To assess the strength of the association between each investigated pair in a systematic way, we adopted the approach of Cohen by using the correlation strength scale[57,58] to interpret the correlation results as follows: values ranging from 0.10 to 0.30 are interpreted as a weak correlation, 0.30 to 0.50 as moderate, and above 0.5 as a strong correlation. We selected Cohen's convention as it is a well-known and standard approach that is used to evaluate the strength of a relationship or effect size.

To support the results obtained from Spearman's correlation and to have a deeper investigation that provides a clearer confirmation regarding the smell proneness of pattern grime instances, a more rigorous statistical test is applied using Odd Ratio (OR).[59] OR is used to evaluate the possibility of pattern grime events to co-occur with smelly pattern instances. OR is calculated based on two groups, and their relationship is illustrated in the following ratio:

$$\mathrm{OR} = \frac{p/(1-p)}{q(1-q)} \tag{1}$$

The first group, denoted as p in the numerator, represents the pattern grime sample, and the second group, denoted as q in the denominator, represents the non-grime sample. The occurrence of each group is considered with smelly and non-smelly instances. If the result of OR is equal to one, this means the probability of each group occurring with smells is equal. An OR value greater than one indicates that the first group (pattern grime sample) is more probable to be associated with smelly instances. On the contrary, an OR less than one means that the second group (non-grime sample) is more likely to have smells.

To answer RQ2, we are mainly interested in identifying the relationships that strongly associate specific pattern grime with individual code smells. Since each project contains different code smells and grime values, we perform the analyses when the data from all five selected projects are combined. The dataset is encoded in binary terms, where "1" denotes the presence and "0" denotes the absence of a specific grime metric co-occurring with a particular code smell within the same pattern instance. This binary representation simplifies the analysis by focusing on the co-occurrence rather than the magnitude of code smells and grime values. In other words, the definition of co-occurrence is the simultaneous presence of code smells and grime within the same pattern instance, without consideration for the accumulation of grime or the frequency of the code smells in that instance. Thus, the co-occurrence in our analysis context can be seen as a binary measure that simply notes the presence or absence of the investigated attributes together (i.e., code smells and grime) rather than quantifying them.

To extract this kind of association, we applied association rules that describe the dependencies between the attributes in a given dataset. Specifically, we employed the Apriori association mining rule algorithm[60] using the Weka tool.[††] Apriori algorithm is a good fit for analyzing Boolean data, where each item is either present or absent. This aligns perfectly with our dataset's structure, which focuses on the presence or absence of grime metrics and code smells. Apriori algorithm also automates the discovery of associations, which can handle large datasets efficiently, a task prone to error when done manually. The algorithm systematically tries different combinations of items (in our case, grime metrics and code smells) in an iterative manner, ensuring a comprehensive analysis that is not biased by manual selection. Further, the rules generated by the Apriori algorithm are straightforward, which makes them easy to interpret. In addition, the Apriori algorithm is commonly selected for association rule mining, particularly in studies examining the relationship between design patterns and code smells,[61] as well as in research aimed at extracting rules for the detection of code smells.[62] This highlights the algorithm's relevance and applicability in software engineering research. These reasons collectively justify the adoption of the Apriori algorithm in this study.

An association rule consists of two sides in which the left-hand side is called the antecedent, and the right-hand side is called the consequent. Both antecedent and consequent represent an attribute in a given dataset. Our dataset has 16 attributes representing the six-grime metrics and the 10 investigated code smells. The antecedent is the pattern grime in this study, whereas the consequent is the code smell. A Boolean representation (i.e., 0 and 1) is used to denote the presence or absence of pattern grime and code smell in each pattern instance in our dataset, as mentioned above. To evaluate the significance of each extracted association rule, two metrics are widely used: Support and Confidence.[60] The Support metric measures the importance of an extracted rule with respect to the whole dataset, whereas the Confidence metric measures how accurate an extracted rule is. Values of Support and Confidence vary between 0 and 1, where higher values give higher significance to the evaluated rule. However, Support and Confidence are not reliable measures when the dataset size is relatively small, as in our case.[63] Therefore, we

alternatively used the Conviction metric[63] to have a more accurate evaluation as this metric combines Support and Confidence in a ratio, as shown in Equation (2).

$$\text{Conviction}(\text{pattern\_grime} \rightarrow \text{smell}) = \frac{(1 - \text{Support}(\text{smell}))}{(1 - \text{Confidence}(\text{pattern\_grime} \rightarrow \text{smell}))} \quad (2)$$

Conviction values vary between 0.5 to ∞ and can be interpreted as follows: 1 indicates that the antecedent and consequent of a given rule are independent, and values greater than 1 indicate possible positive dependency between the antecedent and the consequent. In contrast, values less than one indicate a possible negative dependency between the antecedent and the consequent. To allow the weak association rules to be extracted from our dataset, the minimum values were set for support and confidence in the Weka tool. Minitab[‡‡] is a statistical tool that was used to analyze the collected data statistically and test its normality from normality probability plots. The independent variable in this study is the growth of pattern grime in the selected projects measured by the following pre-defined six metrics: ***cg-naa***, ***cg-napm***, ***mg-ca***, ***mg-ce***, ***og-ca***, ***og-np***, whereas the dependent variable is the 10 investigated code smells.

The last research question, RQ3, is very similar to RQ1 in terms of assessing the presence and the strength of the correlation (if any) but considering different types of design patterns as a dependent variable instead of code smells. Therefore, we followed the same approach and statistical analysis we used to answer RQ1. Particularly, we investigated the number of instances of each identified design pattern type versus each grime metric computed for the same pattern instances in a pairwise manner (e.g., ***<No. design pattern instance of type x, cg-naa in pattern instance x>***) to compute the correlation coefficient for each pair. We considered all identified types of design patterns from the selected projects that belong to the three categories (i.e., creational, structural, and behavioral) listed in Table 3 in the analysis domain, along with all categories of grime metrics. It is worth mentioning that in RQ1, more than one code smell type can be detected from one pattern instance at the same time. However, in RQ3, any instance is classified into only one type of design pattern. Therefore, in RQ3, we conduct the correlation analysis based on the combined data of all projects. Table 5 represents a summary of the statistical tests applied in this study to provide answers to each research question. The null and alternative hypotheses being tested for RQ1, RQ2, and RQ3 are listed below, respectively:

- H_1: *There is no significant relationship between the presence of code smells and the growth of pattern grime*.
- H_{1a}: There is a significant positive relationship between the presence of code smells and the growth of pattern grime.
- H_{1b}: There is a significant negative relationship between the presence of code smells and the growth of pattern grime.
- H_2: *There is no significant association between the group of code smells that occurred with the growth of a specific pattern grime category*.
- H_{2a}: There is a significant positive association between the group of code smells that occurred with the growth of a specific pattern grime category.
- H_{2b}: There is a significant negative association between the group of code smells that occurred with the growth of a specific pattern grime category.
- H_3: *There is no significant relationship between the accumulation of pattern grime and the presence of the different types of design patterns*.
- H_{3a}: There is a significant positive relationship between the accumulation of pattern grime and the presence of the different types of design patterns.
- H_{3b}: There is a significant negative relationship between the accumulation of pattern grime and the presence of the different types of design patterns.

TABLE 5 Summary of statistical tests used to analyze the data and provide answers to achieve research objectives.

Research objectives	Test type	Test objective
Objective 1 (RQ1): Empirical evaluation of the potential relationship between the accumulation of pattern grime and the presence of code smells.	Spearman's rank correlation coefficient[56]	Assess the strength of the relationship between code smells and pattern grime.
	Odd Ratio (OR)[59]	Support the results obtained from Spearman's rank correlation by computing the probability of pattern grime events occurring along with code smells.
Objective 2 (RQ2): Empirical evaluation of the significance of individual code smell when it appears in a specific pattern grime category	Apriori Algorithm[60]	Generate the association rules analysis that assesses if a specific type of code smell has a high probability of co-occurring at a specific level of grime. (i.e., to assess the smell proneness of each pattern grime category against each type of our investigated code smells).
Objective 3 (RQ3): Empirical evaluation of the potential relationship between the accumulation of pattern grime and the presence of design patterns	Spearman's rank correlation coefficient[56]	Assess the strength of the relationship between design pattern and pattern grime.

4 | EMPIRICAL STUDY RESULTS

This section provides answers for each research question supported with empirical evidence by summarizing the statistical test results. To provide general characteristics of the selected projects, Table 6 represents descriptive statistics for each pattern grime variable per each selected project by representing the summation of all grime values, the minimum and maximum grime values, the average of all grime values, and their standard deviation. It also shows the total number of classes in each project, and the number of all possible co-occurrences of pattern grime and code smells instances. We can notice from Table 6 that the afferent coupling (***mg-ca***) and efferent coupling (***mg-ce***) have the highest values compared with the other grime metrics, which could indicate the presence of bad practices as these two metrics assess how much the pattern instances are dependent on the other instances. On the other hand, (**og-np**) metric that assesses the growth of grime between different packages of a given project has the least descriptive statistics values, which is intuitive as the pattern instances usually grow within the same package boundary and rarely grow outside. From Table 7, comparing the number of smelly instances when they contain grime and when they do not, we can observe that smelly-grime instances range from 6 to 110 over the selected projects, whereas smelly-non-grime instances only range from 8 to 25. Another general insight observed from Table 7 is that JHotDraw has the highest number of code smells (i.e., 474) and encountered the highest share of smelly-grime instances (i.e., 110).

TABLE 6 Descriptive statistics of pattern grime values identified for each project.

Pattern grime	Project	SUM	Min	Max	Mean	STD
cg-na	QuickUML	134	0	32	3.19	5.46
	MapperXML	203	0	13	3.19	3.89
	Nutch	983	0	58	7.42	11.85
	JhotDraw	734	0	17	3.00	3.88
	Jrefactory	604	0	22	1.76	3.26
cg-npm	QuickUML	151	0	15	3.60	4.07
	MapperXML	643	0	32	10.31	7.94
	Nutch	908	0	36	6.82	6.31
	JhotDraw	4572	0	51	18.56	16.33
	Jrefactory	8866	0	86	25.81	35.34
mg-ca	QuickUML	1118	0	60	11.89	19.32
	MapperXML	3787	1	44	20.88	12.64
	Nutch	2928	0	38	9.40	9.03
	JhotDraw	26,060	0	141	39.95	43.98
	Jrefactory	39,867	0	166	65.62	49.31
mg-ce	QuickUML	1692	5	42	18.00	12.48
	MapperXML	2819	1	33	15.52	8.68
	Nutch	9377	2	66	30.06	17.53
	JhotDraw	18,874	2	76	28.91	16.21
	Jrefactory	30,002	0	100	0.00	35.38
og-ca	QuickUML	265	1	5	2.82	1.61
	MapperXML	1780	0	15	9.83	3.33
	Nutch	1805	0	13	5.84	3.88
	JhotDraw	4882	1	14	7.48	2.68
	Jrefactory	8490	1	35	13.96	6.33
og-np	QuickUML	186	1	3	1.98	0.93
	MapperXML	446	1	3	2.45	0.65
	Nutch	671	1	3	2.18	0.74
	JhotDraw	1377	1	3	2.11	0.56
	Jrefactory	1246	1	3	2.05	0.59

Note: SUM, summation of all grime values; Min, minimum value; Max, maximum value; Mean, average of all values; STD, standard deviation.

TABLE 7 The total number of classes in each project and the count of instances illustrating grime and code smells co-occurrences.

Projects	# Classes	# Code smells	# Smelly Grime (SG)	# Smelly non-Grime (SnG)	# Non-smelly Grime (nSG)	# Non-smelly non-grime (nSnG)
QuickUML	217	28	6	8	94	228
MapperXML	234	48	18	15	169	199
Nutch	283	145	25	22	285	258
JHotDraw	380	474	110	25	547	271
Jrefactory	563	415	56	23	556	507
Total (all projects)	1677	1110	215	93	1651	1463

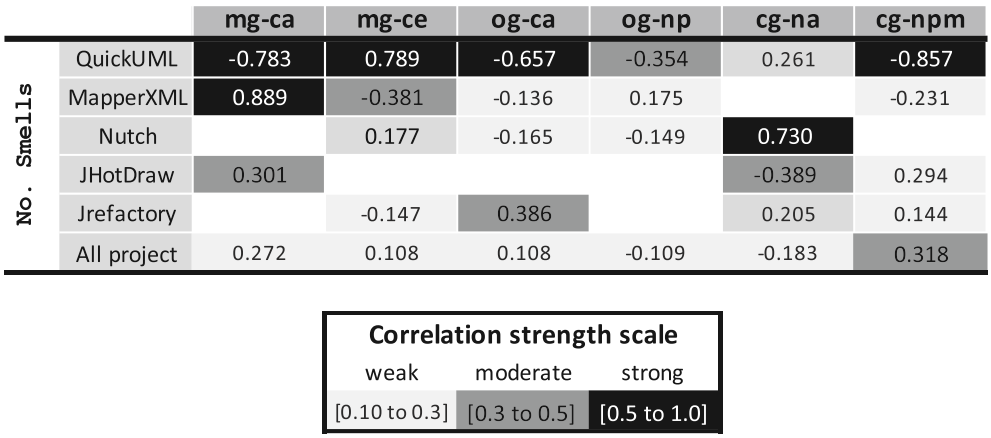

FIGURE 2 Spearman's correlation test results between grime metrics and the number of smells in each project, where the presented correlation strength scale was followed to interpret the result according to the approach of Cohen.[57]

In contrast, QuickUML is associated with the lowest number of code smells (i.e., 28) and consists of a low number of smelly-grime instances (i.e., 6). A similar pattern can be observed for the rest of the selected projects, where projects with a low number of code smells encountered a low number of smelly-grime instances. To verify these general observations, the results of the statistical tests are presented for each research question next.

4.1 | RQ1: What relationship, if any, exists between the accumulation of pattern grime and the presence of the investigated code smells?

To evaluate the relationship between the accumulation of pattern grime and code smells, we calculated Spearman's correlation considering grime and the number of code smells that appear in each pattern instance, as demonstrated in Section 2. Figure 2 reports the result of Spearman's correlation test depicted as a heat map where darker gray shades indicate stronger correlation, insignificant statistical correlation values were not reported, and their places were left blank. It is noticeable from Figure 2 that there is a clear variation in correlation values where each project has different and inconsistent results. For example, the grime metrics in QuickUML show a moderate to strong correlation between grime metrics and the number of code smells. In contrast, cg-np metric that assesses the grime accumulation at the class level shows a strong correlation (i.e., 0.73) with the number of smells in the Nutch project, whereas the other grime metrics reflect a weak correlation. Different correlation results were also obtained when the data of all selected projects were combined, as most grime metrics showed a weak correlation with the number of smells, as shown in Figure 2. Therefore, no clear trend or pattern could be generated based on Spearman's correlation test results. However, the clear variation of the correlation values suggests that the nature of the relationship between the number of smells and grime metrics is project-dependent. In alignment with our findings, the same observation was reported by Izurie and Bieman,[4] where they noticed that each investigated project behaves differently in terms of the accumulation of grime in each grime category, and accordingly, different values were reflected for quality indicators.

To further investigate the relationship between pattern grime and code smells and provide a clearer interpretation, we computed the OR based on two groups: (1) smelly and non-smelly instances when pattern grime is encountered versus (2) smelly and non-smelly instances when

TABLE 8 Statistical results of Odd Ration (OR) test for each project individually and for all projects combined.

Projects	OR	Smell event
QuickUML	1.82	>1
MapperXML	1.41	>1
Nutch	1.03	>1
JhotDraw	2.18	>1
Jrefactory	2.22	>1
Total (all projects)	8.66	>1

TABLE 9 Data for the co-occurrence of individual grime with individual smells.

	Smells										
PatternGrime	GC	FE	BM	IC	FH	SS	EC	BC	RPB	DC	Total[a]
mg-ca	105	61	77	124	8	1273	5	7	8	11	1679
mg-ce	105	61	77	124	8	1273	5	7	8	11	1679
og-ca	105	61	77	124	8	1273	5	7	8	11	1679
og-np	105	61	77	124	8	1273	5	7	8	11	1679
cg-na	108	62	76	123	11	1276	8	10	11	14	1699
cg-npm	108	62	76	123	11	1276	8	10	11	14	1699

[a]Smelly-grime instances.
Abbreviations: BC, Brain Class; BM, Brain Method; DC, Data Class; EC, Extensive Coupling; FE, Feature Envy; FH, Futile Hierarchy; GC, God Class; IC, Intensive Coupling; RPB, Refused Parent Bequest; SS, Shotgun Surgery.

pattern grime is not encountered. The results of the OR test are illustrated in Table 8. It can be observed from the last column in Table 8 that all investigated projects show significant statistical results (i.e., OR > 1, with *P*-value < 0.05), which can be interpreted as the smell events are more likely to be associated with the grime instances group. The same conclusion can be made when the data of all projects are combined. Based on the results reported for Spearman's correlation test and OR test, we can accept the alternative hypothesis (H_{1a}) and conclude that there is a positive relationship between the presence of code smells and pattern grime such that the grime pattern instances are more smell prone than the non-grime pattern instances. From a practical point of view, the findings are intuitively anticipated, as grime and smells are both indicators of bad practices. Therefore, a change in one of them would trigger a change in the other.

Summary: The Spearman's correlation test did not reveal clear trends. However, the variation in correlation coefficients indicates that the connection between the number of code smells and grime metrics may vary from one project to another. To dig deeper into how pattern grime and code smells are related, we calculated the Odds Ratio. The findings from the Odds Ratio analysis indicate a positive association where instances of grime patterns tend to have more code smells than instances without grime patterns. This trend is statistically significant across all projects we examined, evidenced by an Odds Ratio greater than 1 combined with a *P*-value below 0.05. This suggests that code smells are more likely to occur in conjunction with grime patterns.

4.2 | RQ2: Are there particular code smells more frequently appear in a particular category of pattern grime?

As we noticed from the results interpretation of RQ1, the growth of grime and the number of smells is project-dependent; therefore, to eliminate the impact of projects as a nuisance factor, we performed the analysis using the Apriori algorithm[60] on the data of all projects combined into one dataset.

We aggregated the data before applying the Apriori algorithm to provide a baseline understanding of data distribution. Table 9 demonstrates the aggregated number of instances where each grime present with a smell is represented as (1,1) in the dataset and counted as one co-occurrence. In total, the dataset contains 1699 smelly-grime instances. Several observations can be derived from the data presented in Table 9:

- It is clearly noted that the number of co-occurrences with various code smells is identical for most of the grime metrics (mg-ca, mg-ce, og-ca, og-np). This suggests a uniform distribution across these grime metrics, implying that there is no differential impact on the presence of code smells within the context of these metrics. However, there is a notable exception with the cg-na and cg-npm grime metrics, which show a slightly higher number of co-occurrences with all types of code smells, as indicated by their total counts (1699 vs. 1679 for the others). This difference suggests that these particular grime metrics (cg-na and cg-npm) might have a higher impact or a stronger association with the presence of code smells compared with the others.
- Extensive Coupling, Brain Class, Refused Parent Bequest, and Data Class smells recorded the lowest co-occurrence with grime in the same instances in the dataset, with 8, 10, 11, and 14 co-occurrences, respectively.
- God Class, Intensive Coupling, and Shotgun Surgery were the most common smells that collocated simultaneously with grime.
- Shotgun Surgery smell has significantly higher co-occurrence than the other smells as it collocated with grime in 1276 instances out of 1,699.

Since the objective of RQ2 is to investigate if a specific type of smell has a high probability of co-occurring at a particular level of grime, we decided to evaluate the rules that have pattern grime on the left-hand side (as the antecedent) and smells in the right-hand side (as the consequent), separately. Rules that contained smells as antecedent or pattern grime as a consequence were ignored. The obtained association rules are equal to $6*10 = 60$; most of them exhibit weak association, and only two rules show significant statistical results with conviction values greater than one at high confidence level > 95%. Table 10 lists the significant association rules resulting from applying the Apriori algorithm along with its corresponding conviction values. As shown in Table 10, R1 and R2 have conviction values equal to 1.66, indicating a strong positive association between the presence of class grime metrics and the presence of Shotgun Surgery smell in pattern instances. The strong positive association between the presence of class grime and Shotgun Surgery smells could be attributed to the fact that this kind of smell occurs at the structural level of the code. For example, in our dataset, Shotgun Surgery appears at the method level, and both class grime metrics (cg-na and cg-npm) assess grime accumulation at the method level in each pattern instance. On the other hand, the presence of modular and organizational pattern grime was not associated with any of the investigated smells, which might indicate that the growth of grime outside the class boundaries has an insignificant impact on the presence of investigated smells. Notably, the rules representing negative association (i.e., the presence of grime and the absence of smells) have insignificant support, confidence, and conviction values; therefore, they were not reported in Table 10. This finding indirectly supports the results for RQ1 regarding the nature of the relationship between pattern grime and smells. The results of the Apriori algorithm suggest that *there is only a significant positive association between the growth of pattern grime at the class level and the presence of Shotgun Surgery smell*. However, further analysis is required to demonstrate the significance of possible association with class grime and "Shotgun Surgery" co-occurrence and validate our findings.

Summary: The result of RQ2 showed that the majority of grime metrics are equally likely to be present with each code smell, reflecting a uniform distribution of co-occurrence across the dataset, while highlighting a slight deviation in Class grime (specifically cg-npp and cg-na) that represented a slightly higher co-occurrence rate compared with Organizational and Modular grime. Particularly, the result of the Apriori algorithm suggests that there is a significant positive association between the growth of pattern grime at the class level and the presence of Shotgun Surgery smell with significant conviction values equal to 1.66. To corroborate these observations, additional statistical analysis would be necessary.

4.3 | RQ3: What relationship, if any, exists between the accumulation of pattern grime and the presence of the different types of design patterns?

Figure 3 reports the results for Spearman's correlation test presented as a heat map, where darker gray shades indicate a stronger correlation that shows how strongly certain design patterns correlate with the accumulation of grime in software design. The interpretation of the results is based on the same correlation scale used in RQ1, as explained in Section 3.1, Step 5. Note that insignificant statistical correlation values, with

TABLE 10 Significant association rules with corresponding conviction values resulted from applying the Apriori algorithm.

Rule no.	Rules	Conviction value
R1	cg-npm $\Longrightarrow$ ShotgunSurgery	conv:(1.66)
R2	cg-na $\Longrightarrow$ ShotgunSurgery	conv:(1.66)

P-value > 0.05, were not reported in Figure 3, and their places were left blank. Moreover, the asterisks (*) denote that the correlation values cannot be calculated with high confidence due to a lack of sufficient data as the number of identified instances for those patterns is low in the selected projects, which is the case of Prototype, Bridge, Visitor, and Chain of Responsibility design pattern.

Figure 3 shows that Spearman's correlation analysis results have varying levels of association between certain design patterns and the accumulation of pattern grime. However, a clear trend can be observed where strong positive correlations are noted for several design patterns belonging to different design categories with all grime metrics such as Observer, Decorator, Factory Method, and Proxy. For instance, Observer exhibits the strongest positive correlations among the other patterns, with mg-ca at 0.806 and mg-ce at 0.896, indicating a significant coupling level in incoming and outgoing dependencies. The correlation with alien attributes (cg-naa) and methods (cg-napm) is also significantly strong (0.955 and 0.985, respectively), suggesting an extensive interaction with external methods. On the other hand, Singleton shows a significant negative correlation with efferent couplings (mg-ce) at −0.821 and with alien attributes (cg-naa) at −0.574, which could be attributed to the nature of Singleton itself as this design pattern has fewer dependencies on other classes with high degree of encapsulation. The results of Spearman's correlation give a clear trend but not a conclusive answer for all types of design patterns about the relationship across grime metrics. Particularly, Spearman's correlation test results suggest *a significant positive relationship between the growth of pattern grime at the organizational, modular, and class levels and the presence of* Observer, Decorator, Factory Method, and Proxy design patterns. The rest of the design patterns showed insignificant correlation results, so we could not build a conclusive conclusion. Therefore, further deeper investigations using larger datasets are recommended for this study's future direction.

Even though Spearman's correlation results showed a strong, statistically significant positive correlation (i.e., with *P*-value < 0.05), which can highlight relationships between the use of design patterns and the accumulation of grime, the analysis behind these relationships can be complex and multifaceted. The obtained result could be attributed to several factors in relation to design patterns:

- **Complexity and level of interaction:** Some design patterns are inherently more complex than others where multiple classes and object interactions are involved, such as Observer and Decorator. Patterns involve a high level of interaction with external components (e.g., through alien attributes and methods), and their behavior by nature leads to higher coupling or imposes more dependencies, which consequently can increase the potential for grime accumulation.
- **Encapsulation and Modularity:** Design patterns that encourage encapsulation and modularity, such as Singleton, may show a negative correlation with certain types of grime, like efferent couplings, because they are designed to control object creation and usage, thereby reducing dependencies.

Design Pattern Category	Design Pattern	Pattern Grime					
		mg-ca	mg-ce	og-ca	og-np	cg-na	cg-npm
Creational	Factory Method	0.684	0.579	0.579	0.579	0.895	0.895
	Prototype	*	*	*	*	*	*
	Singleton	**0.718**	**-0.821**	0.41	-0.363	**-0.574**	**0.658**
Structural	Adapter	**0.667**	**0.667**	**0.872**	0.205	-0.158	**0.616**
	Bridge	*	*	*	*	*	*
	Composite	*	*	*	**0.875**	*	*
	Decorator	**0.751**	**0.751**	**0.751**	**0.845**	**0.845**	**0.751**
	Proxy	**0.588**	**0.588**	**0.706**	**0.588**	**0.824**	**0.588**
Behavioral	Observer	**0.806**	**0.896**	**0.806**	**0.955**	**0.955**	**0.985**
	State		-0.257	0.314		-0.257	
	Template Method	0.4	**0.5**	0.1	0.2	0.1	-0.1
	Visitor	*	*	*	*	*	*
	Chain of Responsibility	*	*	*	*	*	*

Correlation strength scale		
weak	moderate	strong
[0.10 to 0.3]	[0.3 to 0.5]	[0.5 to 1.0]

FIGURE 3 Spearman's correlation test results between grime metrics and the number of design patterns in each category, where the presented correlation strength scale was followed to interpret the result according to the approach of Cohen.[59]

- **Frequency of use:** Commonly used patterns may show stronger correlations with grime accumulation due to their prevalence in several projects. This means that the more a design pattern is used, the more likely it is to contribute to the accumulation of grime, especially if it is not implemented correctly.
- **Implementation Quality:** The variability in how design patterns are implemented across different projects or teams might influence correlation results. Inconsistent implementations can lead to higher grime accumulation if the patterns are not applied properly. On the other hand, a well-implemented design pattern that closely follows the pattern's intent and structure is less likely to accumulate grime.
- **Calculation of grime metrics:** The way in which pattern grime and coupling are measured can affect the results. The specific metrics used and the methodology for collecting their values can influence the strength and direction of the observed correlations.
- **Domain and context of the investigated projects:** The domain and the context in which each design pattern is applied could also justify the obtained results. Certain domains may have requirements that naturally lead to more complex interactions and dependencies, influencing the amount of grime that would accumulate.

Summary: The analysis for RQ3 reveals that different design patterns correlate differently with the accumulation of pattern grime. Specifically, Observer, Decorator, Factory Method, and Proxy are strongly correlated with high grime metrics, suggesting that they are often involved in complex interactions and dependencies. Such patterns may lead to more coupling and external interaction in the code. Although these correlations are apparent, they are not conclusive for all design patterns, and further research with more extensive data sets is needed to support our findings. The correlations observed could be affected by factors such as the inherent complexity of the design patterns, the extent of their interaction with external components, how frequently they are used, the quality of their implementation, the methodology behind the calculation of grime metrics, and the particular domain and context of the projects studied. These findings raise the need for further advanced analysis on how and why certain design patterns might contribute to the accumulation of grime.

5 | IMPLICATIONS

The investigated relationships between pattern grime, design pattern, and code smells are useful for researchers and practitioners as they can contribute to a deeper understanding of software quality issues. The findings in this paper can help researchers identify which patterns are more prone to grime, guiding them to focus on these areas for developing smell-aware design pattern languages and refactoring techniques based on the Smell-Grime correlation. Researchers can also exploit the findings to develop advanced code analysis tools that can detect and quantify pattern grime, leading to better maintenance strategies. For example, researchers can integrate the findings into static analysis tools to provide warnings when code modifications may introduce smells that historically correlate with pattern grime. Moreover, researchers can use the correlations between specific code smells and pattern grime to create predictive models to anticipate maintenance issues in software projects. Furthermore, the link between grime and smells can be exploited as a hint for smell distribution in the system, which facilitates and accelerates the task of identifying code smells. Consequently, the code reviewers can focus their efforts on finding smells in the identified pattern instances that contain grime, which could improve their productivity by reducing the search space. Similarly, practitioners can use the knowledge of which patterns tend to accumulate grime and the association of smells with grime to prioritize refactoring efforts by focusing on high-risk patterns to reduce technical debt. Practitioners can take proactive steps to improve the quality and maintainability of their software systems based on the insights gained from the analysis of the findings. Practitioners can also use insights from the findings to design code review checklists that look for how coding standards are adopted in design pattern implementation and ensure their use is well-documented.

Although the study is not specifically geared toward real-time applications, its findings can still be exploited by researchers for the design, maintenance, and optimization of real-time systems, especially from the perspective of the challenges that code smells and grime present for performance optimization efforts. Real-time applications often require ongoing optimization to ensure high availability with efficient execution, and these code issues could hinder such efforts.

6 | THREATS TO VALIDITY

This section identifies the threats to the validity of this study and how we mitigated them.

6.1 | Construct validity

We did not make any prior expectations about the results to avoid any bias, and all findings were derived from the statistical analysis and experimental observations. However, the process of data collection plays a vital role in the result. In this study, code smells, design patterns, and pattern grime are detected using only a single tool for each. The reason behind using a single tool is due to the limited availability of such tools and their limited capability as well. To mitigate the impact of this threat, we randomly selected 50 instances (10 instances from each project) to manually validate the results obtained by the selected tools for design pattern identification, and no errors were found. Also, all selected tools were used in previous literature, and they have been empirically validated and showed high recall and precision results. Another possible threat is that if smell detectors rely heavily on the same metrics used for identifying grime, there is a risk that the correlation may appear stronger due to the overlap in detection methods rather than an actual relationship between grime and code smells. However, iPlasma uses more than 80 metrics[64] that measure several dimensions, such as complexity, coupling, size, and cohesion. The Spoon-pttgrime tool relies only on six metrics, two for each grime level: class, modular, and organizational level. Therefore, the risk of circular reasoning due to overlapping metrics is mitigated due to the breadth of metrics and dimensions evaluated.

6.2 | External validity

The external validity of this study is strongly affected by the characteristics of the selected five used projects. All selected projects were open-source Java-based projects, which may hinder the generalizability of the results to all other different languages. To mitigate this threat, we selected well-known projects of different sizes (small to medium) and from different domains. However, we acknowledge that considering industrial projects with larger and more diverse samples encompassing other programming languages would increase confidence in the obtained results. Another identified threat is that the six grime metrics used to assess the presence of grime are all estimators, and including new metrics that consider additional subtypes of grime categories may lead to different findings. Moreover, the accuracy of the automated tools in identifying design pattern instances within code is subject to significant challenges, such as identifying false positives or near-instances. The detection of near-instances, where the code resembles all pattern characteristics but lacks a critical element, which the tools are not sophisticated enough to catch. For example, the developer may use all the standard names for classes but forget to generate a critical relationship between two classes for the instance to exist. This shortfall stems from the tools' reliance on syntactic rather than semantic analysis, which makes these tools struggle to understand the deeper intended functionality of the pattern. Improving the algorithms and heuristics these automated tools use can help better distinguish near-instances from true instances. This involves refining the pattern-matching criteria to account for the critical relationships and structural elements that define each pattern. Thus, this study's result could be considered an initial step toward investigating the potential relationship between smells and grime.

6.3 | Conclusion validity

The investigated link between smells and grime was observed in the selected projects and validated using the appropriate statistical tests. However, the results of Spearman's correlation did not provide a clear trend toward the nature of this link, as each project encountered different correlation results. The clear variation in Spearman's correlation results suggested that a relative number of smells in a given project could affect the findings. However, this assumption requires additional verification. Therefore, we supported the statistical analysis of RQ1 with the OR test, which provided a clear conclusion with significant statistical values.

7 | CONCLUSION AND FUTURE WORK

In this paper, we conducted an empirical study to investigate the relationship between the presence of pattern grime and code smells using six-grime metrics considering three levels of grime: class, modular, and organizational, and 10 types of code smells. The analysis was conducted within the context of five open-source Java projects. We also explored the relationship between design patterns and the accumulation of grime that spans across three categories of design patterns: structural, creational, and behavioral.

Regarding the correlation between grime and code smells, the statistical results reported using Spearman's correlation test and OR test showed that, in general, there is a relationship between the presence of code smells and pattern grime such that the grime pattern instances are more smell prone than the non-grime pattern instances. More specifically, the results of applying the Apriori algorithm show that there is a significant positive association between the growth of pattern grime at the class level and the presence of Shotgun Surgery smell. On the other hand, the presence of modular and organizational pattern grime is not associated with any of the investigated smells, which might indicate that the

growth of grime outside the class boundaries has an insignificant impact on the presence of investigated smells. The statistical strength of the association between the presence of grime and the adoption of certain design patterns, as indicated by Spearman's correlation test, points to a tendency for some design patterns to be more prone to grime accumulation. Notably, the Observer, Decorator, Factory Method, and Proxy patterns appear to be particularly vulnerable to grime accumulation. Besides the statistical analysis result, we also provided a multifaceted justification for the observed results, considering several factors that may influence the relationship between design patterns, pattern grime, and code smells, aiming to provide useful insights from different perspectives. This empirical study identifies a few open issues that can be considered as future research directions. One potential research direction is to investigate the effect of code smell correction on the distribution of associated pattern grime instances in the system. Also, more empirical studies can be conducted considering larger projects other than those available in the P-Mart repository with different programming languages to support this study with more refined findings and increase the confidence of the obtained results. Studying the impact of software evolution over time on the accumulation of pattern grime and code smells is another direction researchers could explore. Further, as software systems grow in complexity and scale, and software engineering practices evolve with the advent of AI and other cutting-edge technologies, the need for automated code quality tools with high accuracy and optimized performance increases to keep pace with the rapid advancements. Therefore, there is a compelling need in future research to examine integrating metaheuristic optimization algorithms[65–70] into the software development lifecycle. These optimization algorithms could be used to automate the detection and refactoring of code smells by simulating the process of natural selection to evolve code bases toward higher quality and lower technical debt. Moreover, these algorithms could aid in developing adaptive systems that automatically suggest design pattern replacements or modifications in response to the emergence of pattern grime. The goal is to create tools that not only flag issues but also provide actionable insights and recommendations for improvement, thus supporting developers in writing cleaner, more maintainable, and robust code. It would also be pertinent to examine case studies or instances where real-time systems are affected by code smells and grime. This could involve assessing the impact on system performance through metrics such as response time, throughput, and fault tolerance. Additionally, considering optimized refactoring techniques that depend on predictable behavior for timing guarantees to alleviate the negative impact on real-time applications would be an interesting future direction.

AUTHOR CONTRIBUTIONS

Maha Alharbi contributed to the study's conception and design, data collection and analysis, and manuscript writing. Mohammad Alshayeb contributed to the study's conception and design, data analysis, and manuscript editing.

ACKNOWLEDGMENTS

The authors acknowledge the support provided by the Deanship of Research Oversight and Coordination at King Fahd University of Petroleum and Minerals. We also acknowledge the assistance provided by Mr. Yazan Abdallah during the experimental setup phase of this study.

CONFLICT OF INTEREST STATEMENT

The authors have no competing interest to declare that are relevant to the content of this article.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available in P-MARt at https://www.ptidej.net/tools/designpatterns/index_html#2. These data were derived from the following resources available in the public domain: - P-MARt, https://www.ptidej.net/tools/designpatterns/index_html#2.

ORCID

Mohammad Alshayeb https://orcid.org/0000-0001-7950-0099

ENDNOTES

* https://github.com/search-rug/spoon-pttgrime

† https://github.com/TamerAbdElaziz/DPVIA

‡ https://www.ptidej.net/tools/designpatterns/index_html#2

§ https://users.encs.concordia.ca/∼nikolaos/pattern_detection.html

¶ https://github.com/search-rug/ssap

http://loose.cs.upt.ro/index.php?n=Main.IPlasma

** https://marketplace.eclipse.org/content/infusion-hydrogen

†† https://www.weka.io/

‡‡ https://www.minitab.com/en-us/

REFERENCES

1. Bahill AT, Botta R. Fundamental principles of good system design. *Eng Manag J*. 2008;20(4):9-17. https://doi.org/10.1080/10429247.2008.11431783
2. Khomh F, Guéhéneuc Y-G. Do design patterns impact software quality positively? In: *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE; 2008:274-278.
3. Gamma E, Helm R, Johnson R, Vlissides J, Patterns D. *Elements of Reusable Object-Oriented Software*. Addison-Wesley Reading; 1995.
4. Izurieta C, Bieman JM. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Softw Qual J*. 2013;21(2):289-323. https://doi.org/10.1007/s11219-012-9175-x
5. Izurieta C, Bieman JM. How software designs decay: a pilot study of pattern evolution. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE; 2007:449-451.
6. Griffith I, Izurieta C. Design pattern decay: the case for class grime. In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*; 2014:1-4.
7. Schanz T, Izurieta C. Object oriented design pattern decay: a taxonomy. In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*; 2010:1-8.
8. Folwer M. *Refactoring: Improving the Design of Existing Programs.(1999)*. Google Scholar Google Scholar Digital Library Digital Library; 1999.
9. Fontana FA, Ferme V, Spinelli S. Investigating the impact of code smells debt on quality code evaluation. In: *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE; 2012:15-22.
10. Fontana FA, Ferme V, Marino A, Walter B, Martenka P. Investigating the impact of code smells on system's quality: an empirical study on systems of different application domains. In: *2013 IEEE International Conference on Software Maintenance*. IEEE; 2013:260-269.
11. Cairo AS, Carneiro G d F, Monteiro MP. The impact of code smells on software bugs: a systematic literature review. *Information*. 2018;9(11):273. https://doi.org/10.3390/info9110273
12. Khomh F. Squad: software quality understanding through the analysis of design. In: *2009 16th Working Conference on Reverse Engineering*. IEEE; 2009: 303-306.
13. Khomh F, Di Penta M, Gueheneuc Y-G. An exploratory study of the impact of code smells on software change-proneness. In: *2009 16th Working Conference on Reverse Engineering*. IEEE; 2009:75-84.
14. Alkhaeir T, Walter B. The effect of code smells on the relationship between design patterns and defects. *IEEE Access*. 2020;9:3360-3373.
15. Izurieta C, Bieman JM. Testing consequences of grime buildup in object oriented design patterns. In: *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE; 2008:171-179.
16. Dale MR, Izurieta C. Impacts of design pattern decay on system quality. In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*; 2014:1-4.
17. Feitosa D, Ampatzoglou A, Avgeriou P, Nakagawa EY. Correlating pattern grime and quality attributes. *IEEE Access*. 2018;6:23065-23078. https://doi.org/10.1109/ACCESS.2018.2829895
18. Reimanis D, Izurieta C. Behavioral evolution of design patterns: understanding software reuse through the evolution of pattern behavior. In: *International Conference on Software and Systems Reuse*. Springer; 2019:77-93. https://doi.org/10.1007/978-3-030-22888-0_6
19. Griffith I. D., "*Design Pattern Decay A Study of Design Pattern Grime and its Impact on Quality and Technical Debt*," PhD Dissertation in progress, 2015.
20. Abdelaziz T, Sedky Adly A, Rossi B, Mostafa M-SM. Identification and assessment of software design pattern violations. النشرة المعلوماتية فى الحاسبات والمعلومات. 13-6:(2)1;2019. https://doi.org/10.21608/fcihib.2019.107517
21. Kerievsky J. *Refactoring to Patterns*. Pearson Deutschland GmbH; 2005. https://doi.org/10.1007/978-3-540-27777-4_54
22. Mantyla M. *Bad Smells in Software—A Taxonomy and an Empirical Study*. PhD thesis, Helsinki University of Technology; 2003.
23. Mantyla M, Vanhanen J, Lassenius C. A taxonomy and an initial empirical study of bad smells in code. In: *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings*. IEEE; 2003:381-384.
24. Wake WC. *Refactoring Workbook*. Addison-Wesley Professional; 2004.
25. Kaur A. A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes. *Arch. Comput. Methods Engrg.* 2020;27(4):1267-1296. https://doi.org/10.1007/s11831-019-09348-6
26. Olbrich SM, Cruzes DS, Sjøberg DI. Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems. In: *2010 IEEE International Conference on Software Maintenance*. IEEE; 2010:1-10.
27. Olbrich S, Cruzes DS, Basili V, Zazworka N. The evolution and impact of code smells: A case study of two open source systems. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE; 2009:390-400.
28. Chatzigeorgiou A, Manakos A. Investigating the evolution of bad smells in object-oriented code. In: *2010 Seventh International Conference on the Quality of Information and Communications Technology*. IEEE; 2010:106-115.
29. Zazworka N, Shaw MA, Shull F, Seaman C. Investigating the impact of design debt on software quality. In: *Proceedings of the 2nd Workshop on Managing Technical Debt*; 2011:17-23.
30. Khomh F, Di Penta M, Guéhéneuc Y-G, Antoniol G. An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Empir Softw Eng*. 2012;17(3):243-275. https://doi.org/10.1007/s10664-011-9171-y
31. D'Ambros M, Bacchelli A, Lanza M. On the impact of design flaws on software defects. In: *2010 10th International Conference on Quality Software*. IEEE; 2010:23-31.
32. Dhillon PK, Sidhu G. Can software faults be analyzed using bad code smells?: An empirical study. *Int J Sci Res Publ*. 2012;2(10):1-7.
33. Hall T, Zhang M, Bowes D, Sun Y. Some code smells have a significant but small effect on faults. *ACM Trans Softw Eng Methodol (TOSEM)*. 2014;23(4): 1-39. https://doi.org/10.1145/2629648
34. Caldiera VRBG, Rombach HD. The goal question metric approach. *Encyclopedia Softw Eng*. 1994;528-532.
35. Guéhéneuc Y-G. P-mart: pattern-like micro architecture repository. In: *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*; 2007:1-3.
36. Li W, Shatnawi R. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *J Syst Softw*. 2007;80(7):1120-1128. https://doi.org/10.1016/j.jss.2006.10.018
37. Fontana FA, Maggioni S, Raibulet C. Understanding the relevance of micro-structures for design patterns detection. *J Syst Softw*. 2011;84(12):2334-2347. https://doi.org/10.1016/j.jss.2011.07.006
38. *QuickUML*. https://quickuml.software.informer.com/ (accessed 2022).

39. *MapperXML*. http://mapper.sourceforge.net/ (accessed 2022).
40. *Nutch*. https://nutch.apache.org/ (accessed 2022).
41. *JHotDraw*. https://sourceforge.net/projects/jhotdraw/ (accessed 2022).
42. *JRefactory*. http://jrefactory.sourceforge.net/ (accessed 2022).
43. Smith JM, Stotts D. SPQR: flexible automated design pattern extraction from source code. In: *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings*. IEEE; 2003:215-224.
44. Fontana FA, Zanoni M. A tool for design pattern detection and software architecture reconstruction. *Inform. Sci.* 2011;181(7):1306-1324. https://doi.org/10.1016/j.ins.2010.12.002
45. Dong J, Lad DS, Zhao Y. DP-Miner: design pattern discovery using matrix. In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. IEEE; 2007:371-380.
46. Dietrich J, Elgar C. Towards a web of patterns. *J Web Semantics*. 2007;5(2):108-116. https://doi.org/10.1016/j.websem.2006.11.007
47. De Lucia A, Deufemia V, Gravino C, Risi M. Design pattern recovery through visual language parsing and source code analysis. *J Syst Softw*. 2009;82(7):1177-1193. https://doi.org/10.1016/j.jss.2009.02.012
48. Antoniol G, Guéhéneuc Y-G. Demima: a multilayered approach for design pattern identification. *IEEE Trans Softw Eng*. 2008;34(5):667-684. https://doi.org/10.1109/TSE.2008.48
49. Shi N, Olsson RA. Reverse engineering of design patterns from java source code. In: *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. IEEE; 2006:123-134.
50. Tsantalis N, Chatzigeorgiou A, Stephanides G, Halkidis ST. Design pattern detection using similarity scoring. *IEEE Trans Softw Eng*. 2006;32(11):896-909. https://doi.org/10.1109/TSE.2006.112
51. Feitosa D, Avgeriou P, Ampatzoglou A, Nakagawa EY. The evolution of design pattern grime: An industrial case study. In: *Product-Focused Software Process Improvement: 18th International Conference, PROFES 2017, Innsbruck, Austria, November 29–December 1, 2017, Proceedings 18*. Springer; 2017:165-181. https://doi.org/10.1007/978-3-319-69926-4_13
52. Fokaefs M, Tsantalis N, Chatzigeorgiou A. Jdeodorant: identification and removal of feature envy bad smells. In: *2007 IEEE International Conference on Software Maintenance*. IEEE; 2007:519-520.
53. Vidal S, Vazquez H, Diaz-Pace JA, Marcos C, Garcia A, Oizumi W. JSpIRIT: a flexible tool for the analysis of code smells. In: *2015 34th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE; 2015:1-6.
54. Ganea G, Verebi I, Marinescu R. Continuous quality assessment with inCode. *Sci. Comput. Programming*. 2017;134:19-36. https://doi.org/10.1016/j.scico.2015.02.007
55. Lacerda G, Petrillo F, Pimenta M, Guéhéneuc YG. Code smells and refactoring: a tertiary systematic review of challenges and observations. *J Syst Softw*. 2020;167:110610. https://doi.org/10.1016/j.jss.2020.110610
56. Field A. *Discovering Statistics Using SPSS: Book Plus Code for E Version of Text*. SAGE Publications Limited London; 2009.
57. Cohen LH. *Life Events and Psychological Functioning: Theoretical and Methodological Issues*. SAGE Publications, Incorporated; 1988.
58. Cohen J. *Statistical Power Analysis for the Behavioral Sciences*. Academic Press; 2013. https://doi.org/10.4324/9780203771587
59. Sheskin DJ. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press; 2003. https://doi.org/10.1201/9781420036268
60. Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*; 1993:207-216.
61. Alfadel M, Aljasser K, Alshayeb M. Empirical study of the relationship between design patterns and code smells. *PLoS ONE*. 2020;15(4):e0231731. https://doi.org/10.1371/journal.pone.0231731
62. Juliet Thessalonica D, Khanna Nehemiah H, Sreejith S, Kannan A. Intelligent mining of association rules based on nanopatterns for code smells detection. *Sci Program*. 2023;2023:1-18. https://doi.org/10.1155/2023/2973250
63. Brin S, Motwani R, Ullman JD, Tsur S. Dynamic itemset counting and implication rules for market basket data. In: *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*; 1997:255-264.
64. Cristina M, Radu M, Mihancea F. iPlasma: an integrated platform for quality assessment of object-oriented design. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance*; 2005:77-80.
65. Dhiman G, Kumar V. Spotted hyena optimizer: a novel bio-inspired based metaheuristic technique for engineering applications. *Adv Eng Softw*. 2017;114:48-70. https://doi.org/10.1016/j.advengsoft.2017.05.014
66. Dhiman G, Kumar V. Emperor penguin optimizer: a bio-inspired algorithm for engineering problems. *Knowl Based Syst*. 2018;159:20-50. https://doi.org/10.1016/j.knosys.2018.06.001
67. Kaur S, Awasthi LK, Sangal A, Dhiman G. Tunicate swarm algorithm: a new bio-inspired based metaheuristic paradigm for global optimization. *Eng Appl Artif Intel*. 2020;90:103541. https://doi.org/10.1016/j.engappai.2020.103541
68. Dhiman G, Kaur A. STOA: a bio-inspired based optimization algorithm for industrial engineering problems. *Eng Appl Artif Intel*. 2019;82:148-174. https://doi.org/10.1016/j.engappai.2019.03.021
69. Dhiman G, Garg M, Nagar A, Kumar V, Dehghani M. A novel algorithm for global optimization: rat swarm optimizer. *J Ambient Intell Humanized Comput*. 2021;12(8):8457-8482. https://doi.org/10.1007/s12652-020-02580-0
70. Dhiman G. ESA: a hybrid bio-inspired metaheuristic optimization approach for engineering problems. *Eng Comput*. 2021;37(1):323-353. https://doi.org/10.1007/s00366-019-00826-w

How to cite this article: Alharbi M, Alshayeb M. An empirical investigation of the relationship between pattern grime and code smells. *J Softw Evol Proc*. 2024;36(9):e2666. https://doi.org/10.1002/smr.2666