sustainability

MDPI

Review

Bad Smells of Gang of Four Design Patterns: A Decade Systematic Literature Review

Sara H. S. Almadi [1], Danial Hooshyar [2,*] and Rodina Binti Ahmad [1,*]

[1] Department of Software Engineering, Faculty of Computer Science and Information Technology, Universiti Malaya, Kuala Lumpur 50603, Malaysia; almadi.sarahs@gmail.com
[2] School of Digital Technologies, Tallinn University, 10120 Tallinn, Estonia
* Correspondence: danial.hooshyar@gmail.com (D.H.); rodina@um.edu.my (R.B.A.)

Abstract: Gang of Four (GoF) design patterns are widely approved solutions for recurring software design problems, and their benefits to software quality are extensively studied. However, the occurrence of bad smells in design patterns increases the crisis of degenerating design patterns' structure and behavior. Their occurrences are detrimental to the benefits of design patterns and they influence software sustainability by increasing maintenance costs and energy consumption. Despite the destructive roles of bad smells in such designs, there are an absence of studies systematically reviewing bad smells of GoF design patterns. This study systematically reviews a 10-year state of the art sample, identifying 16 studies investigating this phenomenon. Following a thorough evaluation of the full contents, we observed that the occurrence of bad smells have been investigated in proportion to four granularity levels of analysis: Design level, category level, pattern level, and role level. We identified 28 bad smells, categorized under code smells and grime symptoms, and emphasized their relationship with GoF pattern types and categories. The utilization of design pattern bad smell detection approaches and datasets were also discussed. Consequently, we observed that the research phenomenon is growing intensively, with a prominent focus of studies analyzing code smell occurrences rather than grime occurrences, at various granularity levels. Finally, we uncovered research gaps and areas with significant potentials for future research.

Keywords: GoF design patterns; systematic review; software engineering; bad smell; code smell; grime; software sustainability

Citation: Almadi, S.H.S.; Hooshyar, D.; Ahmad, R.B. Bad Smells of Gang of Four Design Patterns: A Decade Systematic Literature Review. *Sustainability* **2021**, *13*, 10256. https://doi.org/10.3390/su131810256

Academic Editors: Alok Mishra, Coral Calero, Mª Ángeles Moraga, Jari Porras and Deepti Mishra

Received: 25 June 2021
Accepted: 31 August 2021
Published: 14 September 2021

1. Introduction

Currently, many software developers use object-oriented design patterns in their systems design owing to their traits of reusability, maintainability, and extension capability to forthcoming versions. In addition, their application could eliminate the emergence of defects and faults-proneness in their source code [1–3]. Design patterns are considered a result of good programming practice, primarily aiming to provide solutions in resolving common software problems [4]. They encapsulate the experiences of expert developers as quoted by Freeman: "Instead of code reuse, with patterns you get experience reuse" [5]. In 1995, the Gang of Four (GoF) design patterns catalogue was published, which acclaimed to be the most credible design pattern reference in the literature [6]. The authors catalogued 23 design patterns to resolve recurring software design problems, where they were further segregated into three categories: creational patterns, structural patterns, and behavioral patterns [7]. Creational patterns are aimed at creating objects to serve a suitable purpose to the situation, while structural and behavioral patterns aim to identify ways to represent relationships amongst different objects, and capture behavior amongst the collections of objects, respectively [8].

Consequently, the software engineering community widely adopted the GoF design patterns in their system design, where many software researchers have extensively investigated the impact of its benefits on software quality (e.g., [9–12]). Currently, developers

utilize GoF design patterns as solutions for many issues in software source codes [13]. For instance, they were used to indicate refactoring suggestions and fix code smells symptoms [14,15]. Moreover, developers have applied them to eliminate the emergence of software performance anti-patterns in source codes [16].

However, the GoF design patterns' structure and behavior may be damaged by various bad smells, such as code smells and grime, should they be integrated and extended incorrectly [4,17]. The occurrence of grime in design patterns refers to symptoms of built-up unrelated artefacts in classes that play a role in the design pattern. These unrelated artifacts such as methods, attributes or relationships are unrelated to the intended responsibilities of a design pattern [1]. Moreover, the occurrences of code smells in design patterns are considered as symptoms that indicate a serious problem in design pattern source codes, and a violation of design pattern principles which could demand refactoring to remove them [18].

Bad smell occurrence in design patterns is a recent issue in the GoF design patterns' domain, where code smells and grime are considered the most emphasized symptoms of the design pattern bad smells (DBS) [18–21]. They violate design patterns' principles, responsibilities, and realizations; while they are not considered errors, they degenerate GoF design patterns' structure and behavior [2,4]. Consequently, bad smells in design patterns are found to threaten software quality and sustainability by impacting software modularity, readability, reusability, correctness, and testability [1,22]. This, in turn, introduces a dramatic increase in code complexity, defect and change proneness, which impacts the stability, increases the energy consumption of software, and magnifies future maintenance costs (e.g., [17,23–29]).

In the last decade, increasing attention has been paid by software developers and researchers to this phenomenon (e.g., [1,3,4,17]). Several studies have been published, aimed at identifying and addressing the bad smell occurrences (e.g., code smells or grime) in design patterns by establishing different solutions. For example, in 2019, Reimanis and Izurieta [1] proposed a conformance checking approach to detect grime occurrence in design patterns, whereas Sousa et al. [4] developed a design pattern smells tool using the association rule mining approach to detect code smell occurrences in GoF design patterns. Nonetheless, a very small number of studies were conducted with the aim of systematically reviewing the existing body of knowledge. For instance, in 2018, Bruno et al. [20] conducted a systematic mapping study to review code smell occurrences in GoF design patterns. Their review concerned three different elements: the co-occurrence phenomenon, the effect on software quality, and refactoring. They highlighted the main cases that contribute to the code smells co-occurrence phenomenon, such as the improper implementation of design patterns, further identifying the general trends and productivity of the research domain. However, although their review provides the software engineering community with very insightful information and views, it falls short when it comes to the occurrence of bad smells, such as grime and code smells, in GoF design patterns and in synthesizing their relationships with different granularity levels of analysis and different pattern types and categories.

To bridge this gap, this systematic literature review (SLR) aims to comprehensively identify, categorize, and analyze relevant studies, dealing with this phenomenon with respect to DBS granularity levels (DBGL), and the relationship between the GoF design patterns (e.g., categories and types) and DBS. In addressing the research gap, three research questions were identified:

- RQ1: What are the characteristics and objectives of studies dealing with design pattern bad smell occurrences?
- RQ2: What are the types of bad smells occurring in design patterns, and how are they associated with DBGL and design pattern categories or types?
- RQ3: What are the approaches and datasets used to detect DBS occurrences?

This article is structured as follows: second section deals with the methodology used to carry out the SLR, third section describes our findings and results pertaining to the SLR

research questions, fourth section revolves around a comprehensive discussion, and the final sections describe the limitations and our conclusions.

2. Materials and Methods

This research employs guidelines provided by Kitchenham to conduct the SLR, as these guidelines are considered the most reliable method for conducting an SLR in the field of software engineering [30,31].

Initially, as illustrated in Figure 1, a comprehensive review protocol was followed, aiming at minimizing the likelihood of bias in the studies. This consists of research identification, identifying the SLR research questions, conducting the search process by identifying the databases and keywords, specifying the inclusion and exclusion criteria, applying the quality assessment, and extracting the data from the selected studies [31]. Identification of the research has been described in the previous section of this study, while the remaining stages are elucidated in the following subsections.

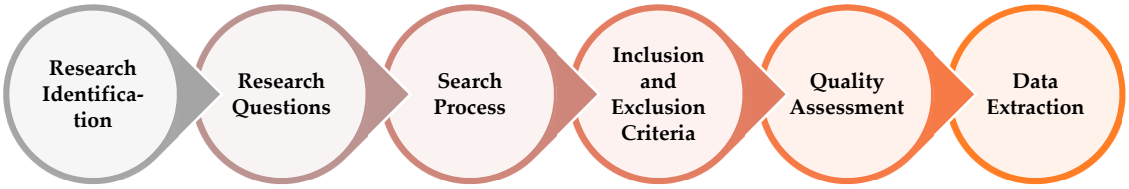

Figure 1. Review protocol used in this research.

2.1. Database and Keywords

The automated and manual mode of searches was conducted in this study. Initially, six leading databases were specified that included information about studies focusing on DBS occurrences: ScienceDirect, IEEE Xplore, Springer, Web of Science, ACM, and Scopus. Based on the identified search keywords, we conducted the automated search in the databases mentioned above (see Table 1).

Table 1. Search strategy employed in this study (Search keywords and Databases).

Year	2010–2020
Search keywords	(“design pattern” OR “GoF design pattern” OR “object-oriented design pattern” OR “Gang of four design pattern”) AND (“decay” OR “grime” OR “smell” OR “bad smell” OR “code smell” OR “defect” OR “software defect” OR “degenerate” OR “change proneness” OR “violation” OR “anti-pattern”)
Databases	ScienceDirect, IEEE Xplore, Springer-Link, Web of Science, ACM, Scopus

The AND/OR Boolean operators were applied with the identified keywords interchangeably in order to retrieve as many results as possible between the years 2010 and 2020. From the word cloud perspectives, the analysis of author-indexed keywords from the selected studies showed that “software design patterns”, “bad smell”, “decay”, and “grime” were amongst the most frequently used keywords (see Figure 2). Additionally, the backward and forward approach was adopted in this study to perform the manual search and to assure the integrity of the systematic search [32]. The aforementioned approach was also used to trace the additional references within the citations of the identified studies.

2.2. Inclusion and Exclusion Criteria

Inclusion and exclusion criteria were developed to identify the relevant studies within the research objectives’ boundaries. These inclusion and exclusion criteria were applied to the retrieved publications in the English language from peer-reviewed conference proceedings and journal articles. However, duplicate studies, and studies produced as book chapters, discussion notes, or reports were excluded from this SLR. The eligibility (inclusion and exclusion) criteria are described in Table 2.

Figure 2. Keywords used in the selected studies.

Table 2. Inclusion and exclusion criteria.

Inclusion Criteria (IC)
IC1: Publication date 2010 to 2020 (both years inclusive).
IC2: Conference proceedings AND Peer-reviewed journal articles.
IC3: In English & accessible online.
IC4: Studies focused on GoF design pattern.
IC5: The studies report at least one bad smell each (e.g., code smell, grime).
IC6: The studies present experimental research and report the results.
Exclusion Criteria (EC)
EC1: The study discusses GoF design patterns and bad smells, but NOT the existence of bad smells within GOF design patterns.
EC2: The study mentions bad smells (e.g., code smell, grime) but Not within the GoF design pattern.
EC3: Studies focusing on the consequences of applying GoF design patterns, but NOT the consequences of bad smells.
EC4: Duplicate studies published in different venues (reporting similar results).
EC5: Studies that use design patterns as a solution.
EC6: Studies that revolves around software development environments for non-professional programmers.

2.3. Study Selection and Data Analysis

The study selection phases and data analysis steps were established to select the most relevant studies corresponding to the SLR objectives. Figure 3 visualizes the study phases and the selection procedures of the SLR. Each article title and abstract were carefully scanned by two independent researchers to review and select the relevant studies. With regards to the search terms appearing in the title and the abstract, some studies were categorized as irrelevant to the phenomenon of interest and were excluded. In the first stage, we excluded 1684 studies, minimizing the number of studies to 119. Subsequently, the full contents of the studies were scanned to finally select 16 studies that were found to have strong relevance to the SLR objectives. In applying the backward and forward approach, five additional studies were included, expanding the selection to a total of 21 studies. Figure 3 details the filtering and selection process of the studies.

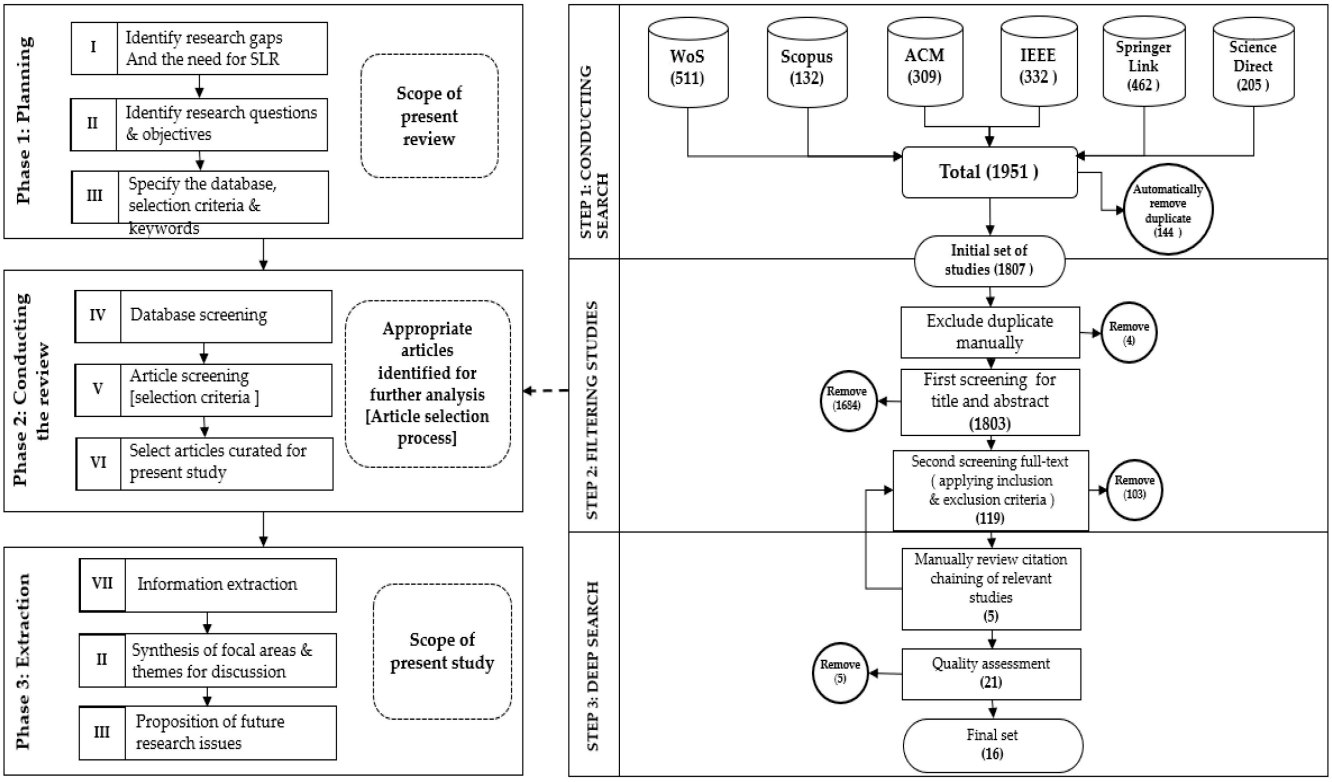

Figure 3. SLR phases and steps involved in the selection of studies.

2.4. Study Selection and Data Analysis

The quality assessment stage is deemed as a very critical stage for affirming the quality of the selected studies in order to criticize their findings and interpretations (e.g., [33,34]). In this stage, we developed five quality assessment criteria questions (QA criteria) to evaluate the remaining studies:

QA1: Are the study objectives and goals clearly defined?
QA2: Does the paper clearly state the research methodology?
QA3: Are the study contributions and limitations clearly stated?
QA4: Is the study data collection process clearly explained?
QA5: Does the study mention how design patterns and bad smells such as code smell and grime relationships were detected?

To evaluate the quality level of the studies, we specified three quality rankings: "high", "medium", and "low", which were used for each QA criterion [34]. The score of 1 was given if the study completely satisfied that quality criterion. Similarly, the score of 0.5 was assigned if the quality criterion was partially satisfied by the study, and we assigned the score of 0 if the quality criterion was not satisfied by the assessed study. In the SLR conducted in this study, based on the five developed QA criterions, the highest possible score is 5 (i.e., $5 \times 1 = 5$), while 0 is the lowest possible score (i.e., $5 \times 0 = 0$). Centered on this coding scheme, the assessed study is considered to have high quality if the score >3, medium quality if the score is $\leq$3 and $\geq$1, and low quality if the assessed study scored <1. Table 3 describes a sample of the quality assessment process for seven studies. In total, 16 studies were categorised as having high to medium quality, with the remaining 5 studies were excluded because they were found to be of low quality (resulting in the inclusion of the 16 studies in the medium to high quality classification).

Table 3. Example of quality assessment criteria.

Study ID	QA1	QA2	QA3	QA4	QA5	Total	Include/Exclude
1	1	1	1	1	1	5	Include
2	1	1	1	1	1	5	Include
3	0.5	1	0.5	0.5	0.5	3	Include
4	1	1	1	0.5	0.5	4	Include
5	0.5	0.5	0.5	0	0	1.5	Exclude
6	1	0.5	1	0	1	3.5	Include
7	1	1	1	1	0.5	4.5	Include

2.5. Data Extraction

The data extraction stage is a critical one that aims to collect the data from the selected studies which have passed the first criteria of quality. A data extraction form was developed for the purpose of recording the extracted data from the 16 studies to ensure the completeness of the data collection stage [33]. This form included elements such as the study ID, reference and year, study objective, publication venue, dataset utilized, bad smell types, detection methods and strategies, design pattern type and category, and the granularity level of the respective analysis, as shown in Table 4.

Table 4. Data extraction.

Elements	Description
ID	The identifier of the study.
Reference, year, and Publication venue	Extraction of the authors name, publication year and the venue of the publication.
Study type	Categorizion of the study (i.e., journal article or conference proceeding).
Study objectives	Identification of the main aim of the study.
bad smells type	Identification of the type of the bad smells which are presented in the study (i.e., code smell, grime).
Design pattern type and category	The type of the pattern discussed in the study (i.e., singleton, adapter, etc.) and the category it belongs to (i.e., creational patterns, structural patterns, or behavioral patterns).
Granularity level of analysis	The level of design pattern investigated in the study (i.e., Design Level, Category level, Pattern level, and Role level).
Detection method/tool/approach/strategy	The proposed detection method or the adopted tool and strategy for the detection of bad smells.
Dataset utilized	The dataset utilized in the study to investigate DBS occurrences.
context of dataset utilized	To identify the context of the utilized dataset (i.e., educational context, gaming context, etc.)

The contents of the final selected studies were carefully reviewed to extract the relevant data for each identified element accurately. In addition to the data extraction elements described in Table 4, the gaps and challenges faced by the researchers of the respective domains were also identified.

3. Results

3.1. Characteristics and Objectives of DBS Studies

Based on our SLR's objectives, 16 studies qualified in satisfying our eligibility criteria, which included seven journal articles and nine conference proceedings. Table 5 delineates an overview of the selected studies and their corresponding responses to the SLR research questions. The checkmark sign is used to indicate a complete response to the research questions by the respective studies. Reassuringly, most of the studies fully responded to our identified research questions, while only six studies did not completely fulfil all three research questions. Furthermore, the demographic characteristics of the selected studies show that the majority of studies originate from the USA, Brazil, and the Netherlands, and these were respectively published by Izurieta et al., Sousa et al. and Feitosa et al.

Table 5. Overview of the related studies and their responses to the SLR research questions.

ID	Reference	Publication Year	Venue	Title	RQ1	RQ2	RQ3
1	[17]	2020	Journal article	Empirical study of the relationship between design patterns and code smells	✓	✓	✓
2	[4]	2019	Conference proceeding	An exploratory study on cooccurrence of design patterns and bad smells using software metrics	✓	✓	✓
3	[1]	2019	Conference proceeding	Behavioral Evolution of Design Patterns: Understanding Software Reuse Through the Evolution of Pattern Behavior	✓	✓	✓
4	[23]	2019	Journal article	Methodology for the quantification of the effect of patterns and anti-patterns association on the software quality	✓		✓
5	[24]	2018	Journal article	Correlating Pattern Grime and Quality Attributes	✓	✓	
6	[25]	2017	Conference proceeding	The Evolution of Design Pattern Grime: An Industrial Case Study	✓	✓	
7	[18]	2016	Journal article	The relationship between design patterns and code smells: An exploratory study	✓	✓	✓
8	[35]	2016	Journal article	Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults	✓		✓
9	[36]	2015	Conference proceeding	Co-Occurrence of Design Patterns and Bad Smells in Software Systems: An Exploratory Study	✓	✓	✓
10	[19]	2014	Conference proceeding	Design pattern decay: the case for class grime	✓	✓	✓
11	[26]	2014	Conference proceeding	Impacts of design pattern decay on system quality	✓	✓	
12	[37]	2013	Conference proceeding	Code Quality Cultivation		✓	✓
13	[2]	2013	Journal article	A multiple case study of design pattern decay, grime, and rot in evolving software systems	✓	✓	✓
14	[27]	2010	Conference proceeding	Object oriented design pattern decay: a taxonomy	✓	✓	✓
15	[38]	2017	Conference proceeding	Evaluating co-occurrence of GOF design patterns with god class and long method bad smells	✓	✓	✓
16	[39]	2018	Journal article	Detecting Software Bad Smells from Software Design Patterns using Machine Learning Algorithms	✓	✓	✓

Navigating into a deeper insight of the objectives and characteristics of the selected studies, we applied the definitions of the four granularity levels of Mohammed and Elfish [40], and Alfadel et al. [17] for analyzing the occurrence of bad smells in design patterns. Figure 4 shows the definitions of the utilized DBGL in analyzing the studies with DBS occurrences (design level, category level, pattern level, and role level). Each selected study was assessed to identify if it was analyzing DBS occurrence in one or more levels of DBGL. We discovered that all 16 studies focused on the pattern level, out of which eight studies focused on the role level, four studies on the design level, and only two studies on the category level (see Table 6).

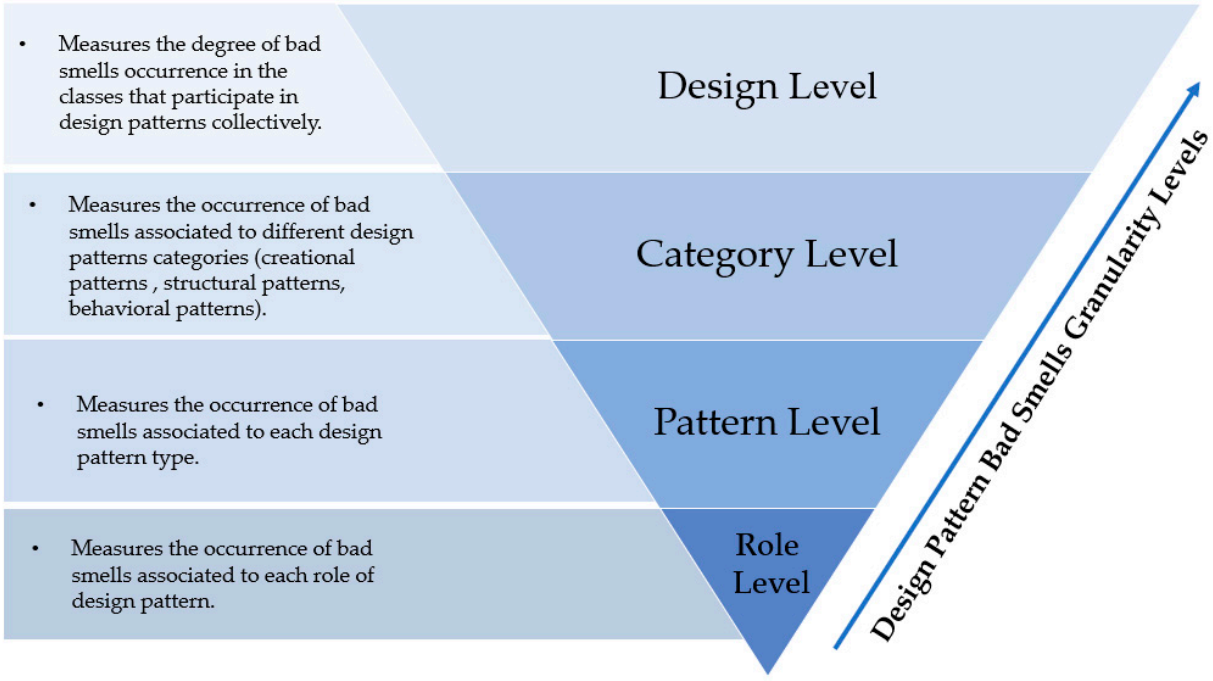

Figure 4. Granularity levels for analyzing DBS occurrences.

Table 6. Summary of DBS granularity levels and types of patterns categories.

DBS Granularity Level of Analysis	Types of Pattern Categories	Number of Studies	Paper ID
Design level	Creational patterns	4	1, 4, 7, 14
	Structural patterns	3	1, 4, 7
	Behavioral patterns	4	1, 4, 7, 14
Category level	Creational patterns	2	1, 14
	Structural patterns	1	1
	Behavioral patterns	2	1, 14
Pattern level	Creational patterns	14	1, 2, 4, 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 3
	Structural patterns	13	1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 15, 16
	Behavioral patterns	16	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,16
Role level	Creational patterns	5	1, 2, 4,15, 13
	Structural patterns	7	1, 2, 4, 12, 15, 3, 13
	Behavioral patterns	8	1, 2, 4, 9, 12, 15, 3, 13

In a more specific perspective, the selected studies investigated DBS occurrences in the three GoF design pattern categories over DBGL interchangeably. However, we discovered that the selected studies commonly analyzed DBS in behavioral patterns over the pattern level. However, the creational patterns, structural patterns, and behavioral patterns were analyzed equally over the design level. Therefore, the studies mainly focused on behavioral patterns and structural patterns over the role level. Astoundingly, the analysis of DBS in behavioral patterns, structural patterns, and creational patterns over the category level was seen to be somewhat neglected and seldom considered.

To reveal the DBS research community's evolving focus over the years, a heatmap has been generated to explore the relationship between DBS study objectives and the selected studies' year of publication. Figure 5 shows a limited number of studies until 2012, focusing on analyzing the DBS occurrence in the category and pattern levels equally. However, the number of studies increased dramatically from 2013 to 2014, with a primary focus on analyzing DBS in the pattern level, followed by the role level. Interestingly, from 2015 to 2018, there was an increase in the analysis of DBS in a variety of DBGL, however, the primary focus remained in the pattern level, and was limited on the design and role levels. In the final two years, from 2019 onwards, the focus expanded in the role level and design level, and was limited in the category level. It was observed that the point of focus throughout the years in analyzing DBS occurrences was in the pattern level, while the category level was noticeably ignored by the research community.

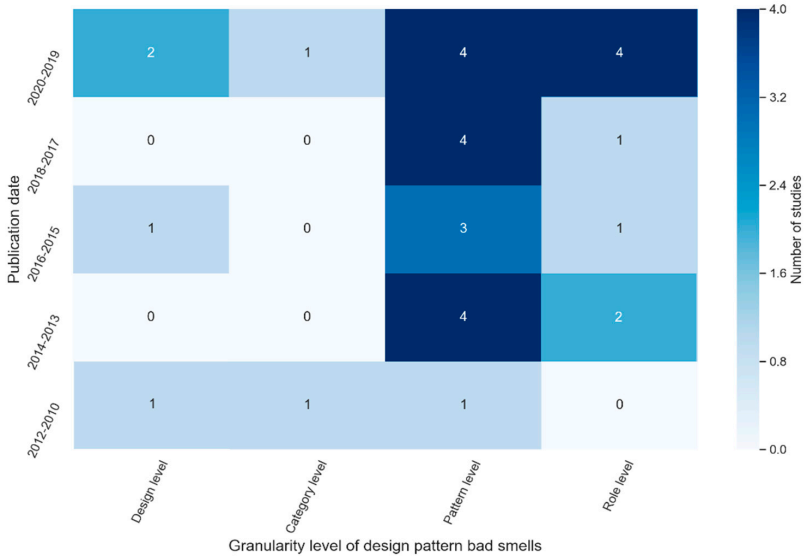

Figure 5. The evolving focus of the DBS research community in the past decade.

Additionally, to understand the objectives of the study in the DBS research domain, we developed heatmaps to discover the association between the studies' objectives and the investigated design pattern types. As the GoF design patterns include 23 design pattern types, divided into three design pattern categories, we developed three heatmaps for each design pattern category, as illustrated in Figure 6.

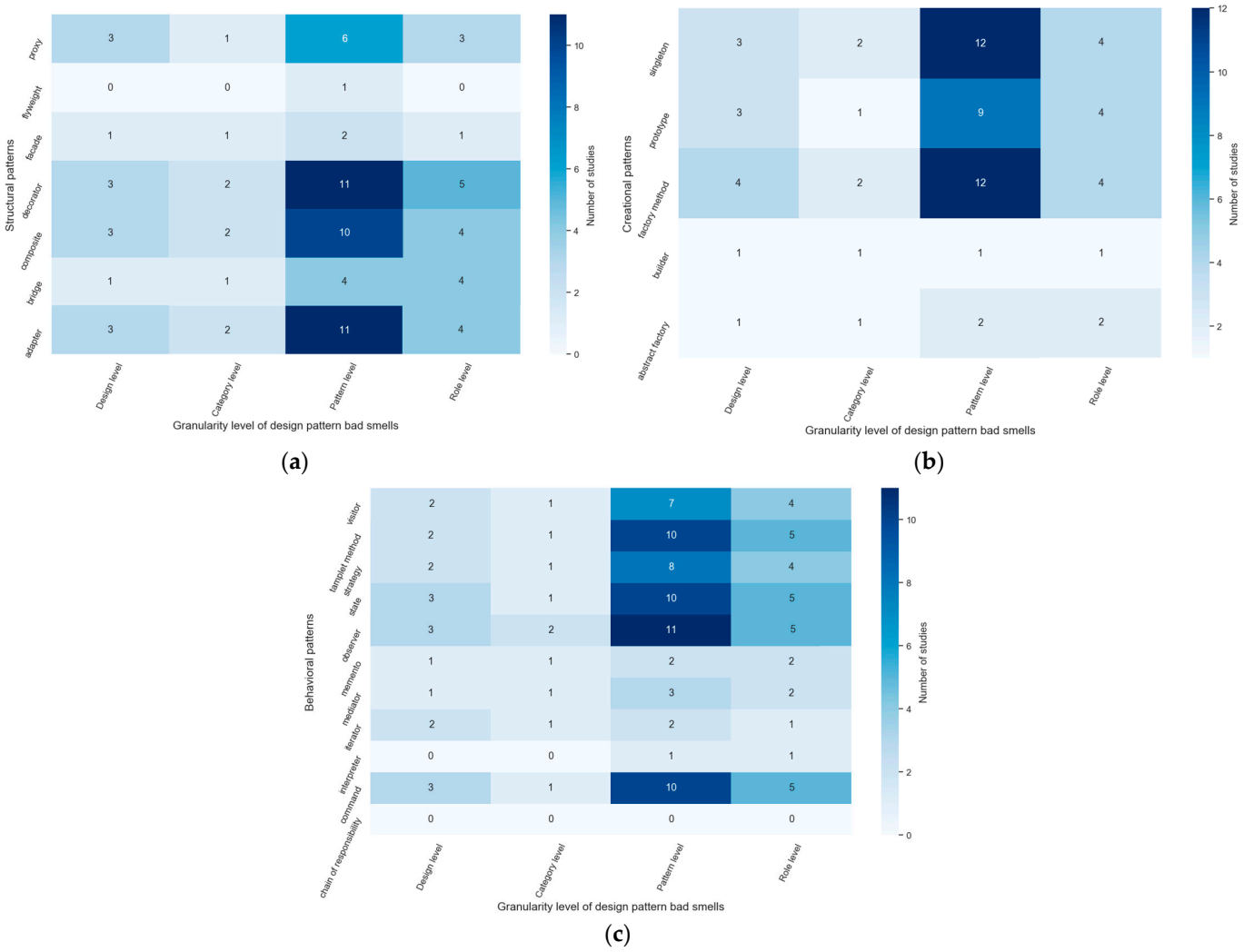

Figure 6. Different characteristics of DBS granularity levels over design patterns types: (**a**) structural patterns, (**b**) creational patterns, and (**c**) behavioral patterns.

Figure 6a demonstrates that DBS in structural patterns have essentially targeted the Decorator pattern, Adapter pattern, and Composite pattern in Pattern level, and narrowly against the Proxy pattern, Bridge pattern, Facade pattern, and Flyweight pattern. However, investigating DBS occurrence in Flyweight pattern was completely neglected at the design level, category level, and role level. The occurrence of DBS in structural patterns thoroughly focuses on the pattern level, in a greater magnitude than in the other three DBGL levels. Figure 6b similarly shows that the research community of the DBS domain also focuses on the pattern level to explore DBS occurrences in creational patterns. The Singleton pattern, Prototype pattern, and Factory method patterns are the most investigated creational patterns at the pattern level and role level. Figure 6c distinctly shows the most prevalent studies that are investigating DBS in behavioral patterns, focusing on the Observer pattern over the pattern level. Similarly, the Template method, State, and Commend patterns gained great attention over the pattern level. Besides, the role level is

considered the second most popular level that gained attention by the research community, primarily associated with the Template method, Strategy, State, and Observer patterns.

3.2. DBS Occurrence Types

Based on the analysis of the DBS studies, we discovered that DBS can be recognized by their bad smell types, their occurrence degree according to DBGL, and their occurrences concerning the GoF design pattern (types and categories). Relying on the 16 selected studies in our research, we explored the methods of characterizing DBS studies based on DBS type, thus leading us towards the analysis of the DBS type against its association with DBGL and GoF design pattern categories and types (more information on DBS types is presented in the Supplementary File).

Pertaining to our investigation of the selected studies, we identified 28 DBS types that occurred in design pattern instances, grouped into two main categories: code smell occurrence and grime occurrence. Code smell occurrences in design patterns are symptoms indicating a violation in pattern principles, or structure. These violations are made by developers during the process of applying or extending design patterns for handling new software requirements. For example, Sousa et al. [4] argued that developers may add many gets and sets methods into a pattern class, not having many features over these data, and leading to an accumulated level of code smells in design pattern. Furthermore, grime occurrence is a symptom indicating a violation in design pattern responsibilities and realization. This happens when developers add artifacts (i.e., relationships, attributes, or methods) into a pattern structure which is not related to the pattern responsibilities which, in turn, results in the degeneration of the pattern's structure integrity and reusability [2].

The research community either focused on detecting and investigating the code smell occurrences or grime occurrences in the GoF design pattern, considering different DBGL of analysis. Thus, nine studies (56.25%) emphasized code smell occurrences in the GoF design patterns, while seven studies (43.75%) emphasized grime occurrence in the GoF design patterns. In comparing the investigation of the two main categories in different DBGL, it showed that the highly emphasized granularity level is the pattern level (see Figure 7).

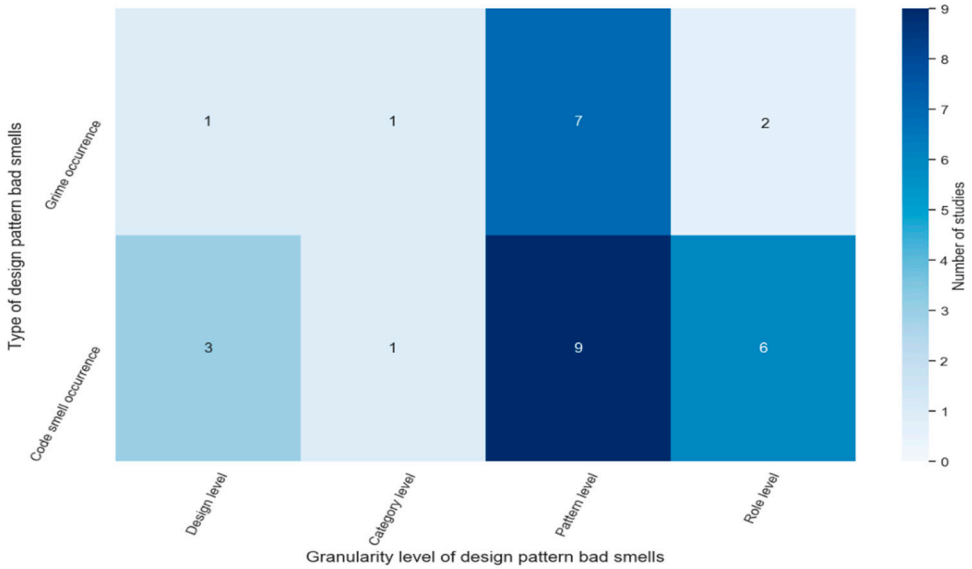

Figure 7. Types of DBS relative to its granularity level.

Therefore, the design level and role levels were mainly used to support the investigation of code smell occurrences in design pattern. The research of grime occurrence in design pattern narrowed down on the pattern level, and slightly on the category level and design level. Surprisingly, the design level and Category level gained less attention throughout the studies.

Figure 8 shows the DBS types, classified under the two main categories (code smells and grime), which are explained in the following sections.

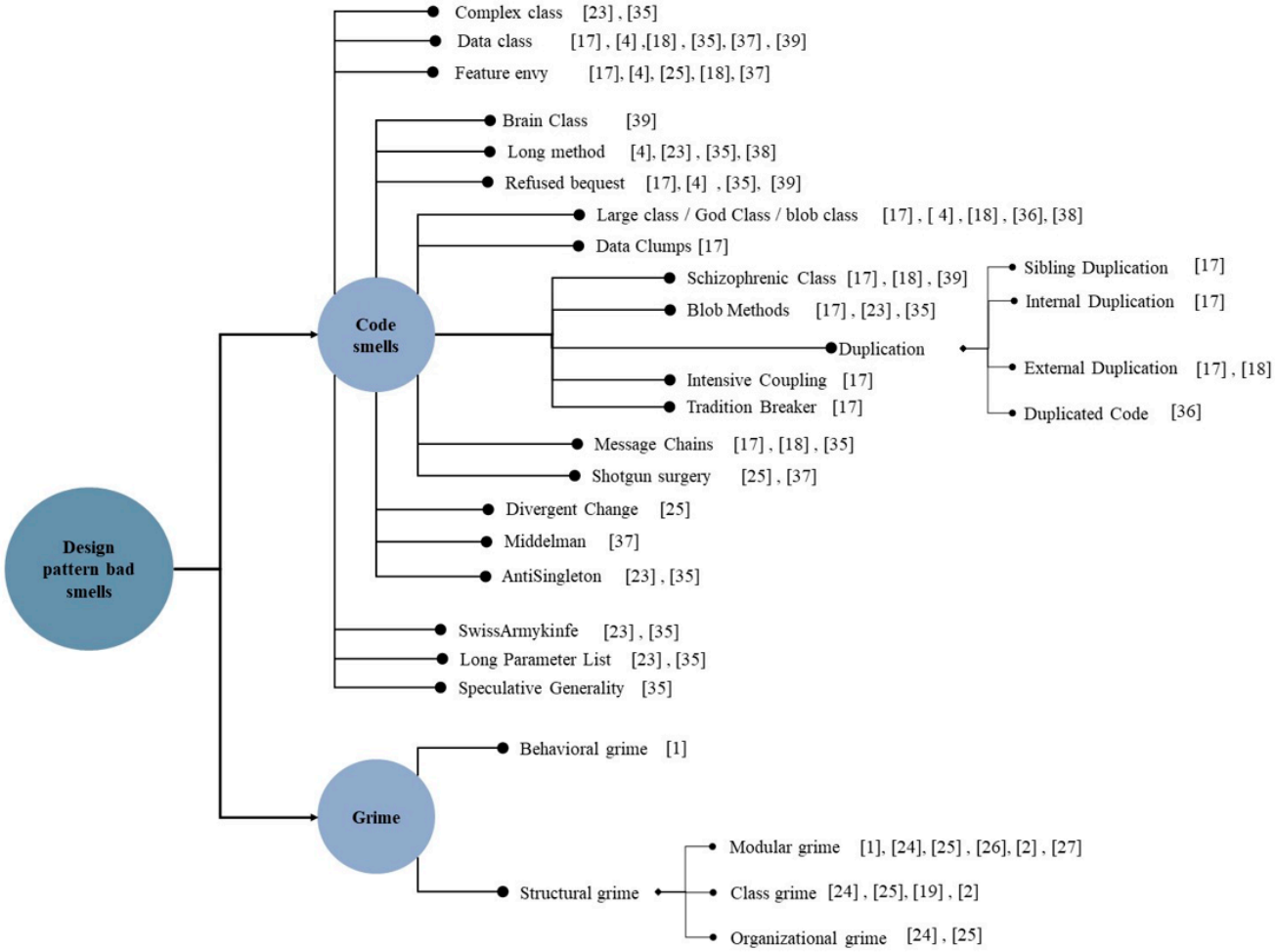

Figure 8. Types of DBS, classified into Code smells and Grime.

3.2.1. Code Smell Occurrences in GoF Design Patterns

Bases on the selected studies, we identified 24 types of code smell occurrences in design patterns. The studies which investigated code smell occurrences in design patterns depend on the phenomenal concurrence by Fowler et al. [41], and the catalogue of code smells by Lanza and Marinescu [42], and Brown et al. [43]. Code smells in design patterns illustrate that the pattern instances suffer from the symptoms of violations in different aspects, either a pattern structural violation, or a principles violation. Each identified type of code smell occurrences in design patterns identify particular symptoms and violations. For instance, the Refused bequest code smell symptom indicates that the features and attributes inherited from the superclass are not used by the subclass (see Appendix A, Table A2). Thus, the occurrence of the Refused bequest is evidence of a wrong inheritance structure [4].

Accordingly, our results illustrated that Feature envy, Data class, Long method, Refused bequest, Large class, God class, and Blob class symptoms are the most investigated code smells amongst the 24 code smell types, where eight studies indicated high occurrences between these code smell symptoms and the design patterns' instance. Besides, Blob methods and Schizophrenic class, Message chain, and Shotgun surgery code smells were emphasized by seven studies interchangeably, while the other code smell types attracted less attention from the DBS research community (see Figure 8 for more details).

3.2.2. Grime Occurrence in GoF Design Patterns

Regarding grime occurrences in design patterns, Izurieta et al. [44] argued that grime occurrences vary according to whether they accumulate or degenerate the structure or behavior of the pattern. Moreover, the grime could occur at the pattern structure level. It can be observed via a static analysis of the source code or design, which could either be extracted into UML class diagrams [45], or analyzed by the deviation of pattern behavior, observed from a flow of information perspective that captures the operational side of a design pattern at run time, which could be reflected by the UML sequence diagrams [46].

Consequently, grime occurrences are grouped into Structural grime and Behavioral grime, as we have specified four types of grime occurrences in design patterns, behavioral grime and three structural grime types, as follows (see Appendix A, Table A1):

(1) Behavioral grime shows a symptom of pattern behavioral deviation which can be measured by improper order of sequences or excessive actions. Improper order indicates that the order of pattern behaviors occur incorrectly, while excessive actions indicates that the pattern behavior shows excessive actions, obstructing the pattern's expected run time [44].
(2) Modular grime, within the Structural grime, is a symptom of increasing the pattern's coupling, which could be tracked by the number of relationships (generalizations, realizations, associations, and dependencies) that the pattern class has with another pattern or non-pattern class [45].
(3) Class grime, within the Structural grime, is considered a symptom of increasing pattern class methods and attributes which are not related to the responsibilities of the pattern [19].
(4) Organizational grime, within the Structural grime, reflects a symptom of increasing the number of pattern files and namespace coupling, which is not involved in the responsibilities of the pattern [24].

Therefore, our results of the selected studies show that most of the studies of grime occurrence in design patterns are focused on Structural grime while only one study focused on investigating Behavioral grime occurrence in the GoF design pattern, which was published recently. Indeed, seven studies investigated the Structural grime occurrence in design patterns, while six studies out of seven focused on examining the occurrence of Modular grime, five studies focused on Class grime and only two studies focused on Organizational grime (see Figure 8 for more details).

3.3. The Association of DBS to Granularity Levels

Concerning our analysis of the 16 selected studies, code smells and grime occurrence studies have analyzed the degree of accumulating DBS over different DBGL. DBGL are a great way to deal with the degree of accumulation of DBS, which provides thorough understanding for the developers to emphasize the implementation of design patterns involving certain levels [17,40]. For example, Reimanis and Azurites [1] claimed, based on analyzing design pattern grime occurrence in Category level, behavioral grime often co-occurs with patterns in the behavioral category as they are prone to behavioral deviations. Based on our findings, amongst all the selected and reviewed studies, Pattern level and Role level were reported frequently (see Table 6). Few studies have used four DBGL for analyzing DBS. In comparison, most studies reported DBS, focusing on one or three DBGL levels, as 7 studies focused on Pattern level and Role level at the same time while a majority of the studies focused on Pattern level only.

In exploring more patterns from the 16 selected studies, we applied a parallel coordinate visualization to understand how DBS (code smells and grime) promote DBGL levels. In Figures 9 and 10, the studies disseminated over DBGL parallel coordinate visualization depend on how often the studies revealed substantial evidences of investigating DBS on different DBGL (this plot demonstrates the coloring of the polylines dependencies based on the numerical values). Two parallel coordinate visualizations have been created, one

for visualizing code smell studies, and the other for visualizing grime studies to provide a more profound insight into the distinct trends between the various DBS and DBGL.

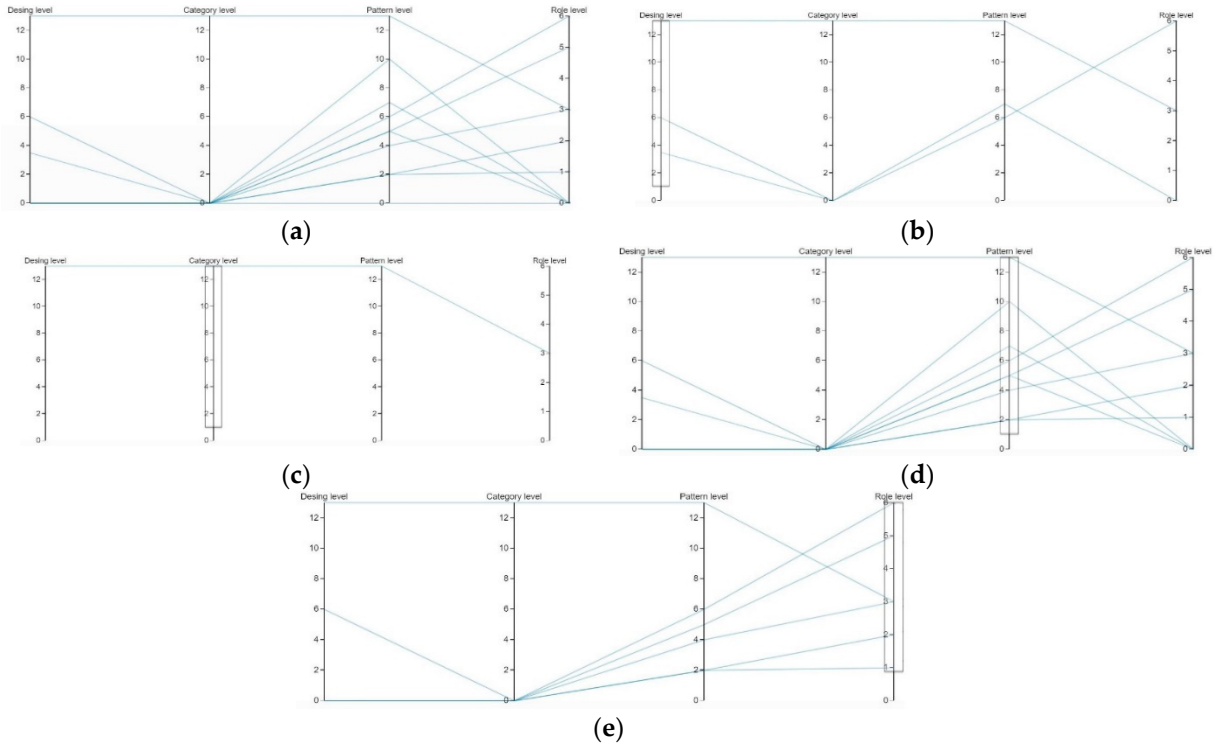

Figure 9. Distribution of code smell occurrence studies amongst the four types of DBS granularity levels of analysis: (**a**) all code smell occurrence studies; (**b**) code smells studies scored high and medium in the design level of analysis; (**c**) code smells studies scored high and medium in the category level of analysis; (**d**) code smells studies scored high and medium in the pattern level of analysis; and (**e**) code smells studies scored high and medium in the role level of analysis.

To initialize the visualization, each design pattern DBGL is scored separately by assigning weightages, followed by applying parallel coordinates to visualize each phase. Explicitly, scores of 0, 0.5, and 1 were assigned to each DBGL. Score 0 was assigned if the study was not shown as part of a particular level in DBGL in the context of DBS types. Score 0.5 was assigned if the study showed to be a part of DBGL, and score 1 was used if the study investigated DBS occurrence using DBGL, while providing evidences of the DBS occurrences' impact within the granularity levels. The maximum score identified for parallel coordinates visualization in Figure 9 is 24, if all design pattern code smell types investigated are within the four DBGL. The maximum score identified for a parallel coordinate visualization in Figure 10 is four, if all design pattern grime types investigated are within the four DBGL.

Figure 9b–d present the same distribution, limited to the code smells studies that gained high or medium score in the design, category, and pattern levels, respectively. We observed that only one study could obtain a score of more than 12 in the design, category, or pattern levels. However, Figure 9e shows various trends as the distributions are limited to the studies which achieved a score of 6 at the role level. More specifically, to analyze the parallel coordinates, we considered the studies which achieved a score of 1 or more than 1, as a medium and high score respectively. As illustrated in Figure 9b, 3 studies scored medium or high in design level, whereas at the pattern and role levels, studies 1, 9, and 6 earned high or medium scores (see Figure 9c–e). Particularly, of the three studies which earned a high or medium score at the design level, they also earned a high or medium score at the pattern and role levels.

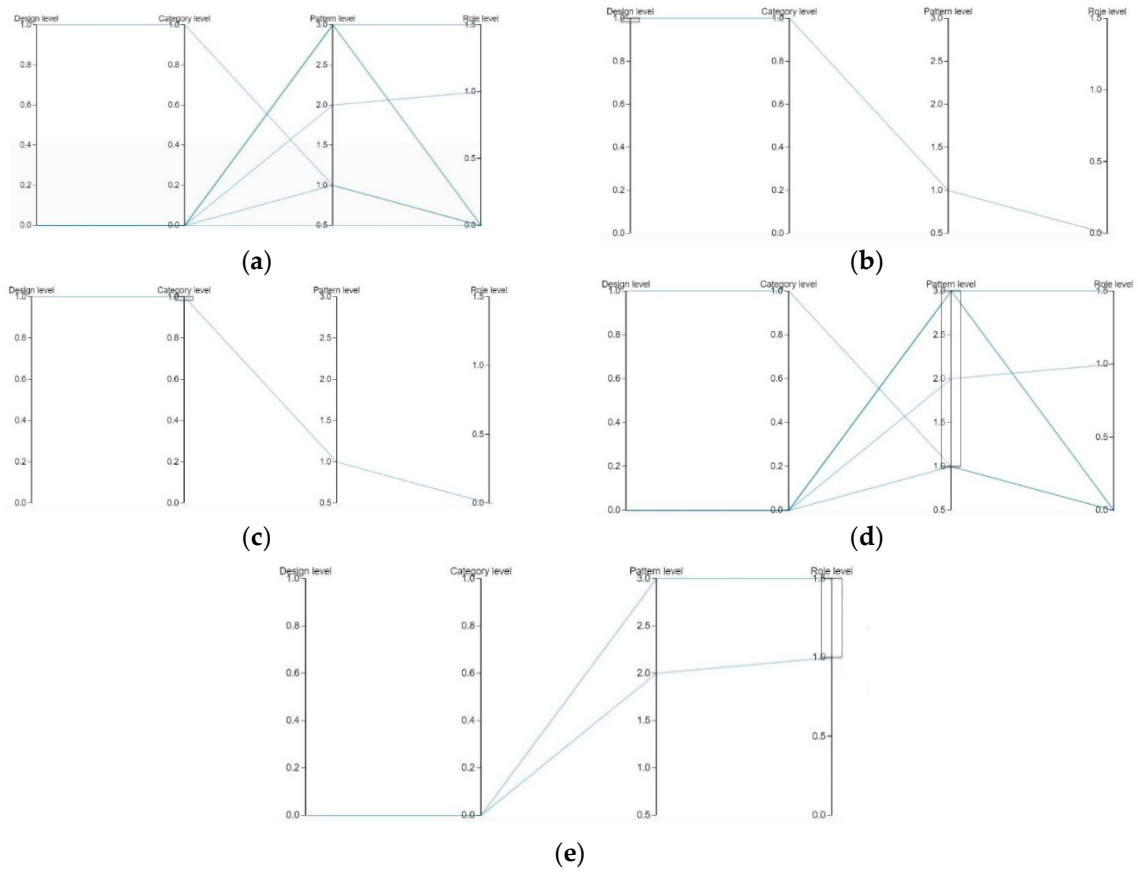

Figure 10. Distribution of grime occurrence studies among the four types of DBS granularity levels of analysis: (**a**) all grime occurrence studies; (**b**) grime studies scored high and medium in the design level of analysis; (**c**) grime studies scored high and medium in the category level of analysis; (**d**) grime studies scored high and medium in the pattern level of analysis; and (**e**) grime studies scored high and medium in the role level of analysis.

Furthermore, only one study out of nine, or three studies that acquired a score of high or medium at the pattern or role levels, which could also achieve a score of high or medium in both the design and category levels, respectively. More specifically, studies 9 and 7 scored more than 0 at the pattern and role levels. Therefore, studies 1 and 2 achieved a score of more than 0 at the design and category levels. These findings indicate that less attention has been given towards the design and category levels than to the other two levels. Moreover, Figure 9a–e demonstrate that only one study achieved a medium and high score in the four granularity levels of DBGL at the same time. Thus, it is noticeable that the majority of the studies ignore analyzing the occurrence of code smells in design patterns over four granularity levels of the DBGL simultaneously.

Figure 10b,c have a similar distribution limited for grime occurrence studies as only one study could achieve a score of 1 at the design and category levels. However, Figure 10d,e which offers distributions limited to the selected studies, achieved a score of 3 and 1.5 at the pattern and role levels, respectively. More specifically, one study earned a medium or high score at the design and category levels (see Figure 10b,c). Otherwise, 6 and 2 studies attain medium and high score at the pattern and role levels (see Figure 10d,e). More particularly, one study which acquired a medium score at the design level could also obtain a score of medium or high at the category and pattern levels. Explicitly, studies 7 and 2 scored more than 0 at the pattern and role levels, respectively. Therefore, one study gained a score of more than 0 at the design and category levels. These findings clarify that the design level and category level have received less attention compared to the other two levels. Moreover, according to Figure 10a–e, it appears factual that no study could achieve a score of medium or higher in all DBGL at the same time. Noticeably, the studies fall short

in their ability to properly associate their investigation and analysis of grime occurrence in design pattern at the category and design levels, and are seen to only focus at the pattern and role levels.

To comprehensively understand the extent of focus of the studies in analyzing code smell occurrences in design patterns and their association to the different granularity levels, we applied a heatmap to reveal the relationship between the identified 24 code smells and the four DBGL (see Figure 11). It was discovered that the granularity level that is most frequently associated with the occurrence of code smells in design patterns is the pattern level. Feature envy, Blob class, and Data class were frequently associated with the pattern, role, and design levels of analysis, sequentially. We discovered that both the pattern and role levels appear to be important elements concerning code smells in all types of code smell occurrences, while being marginally focused on the category level.

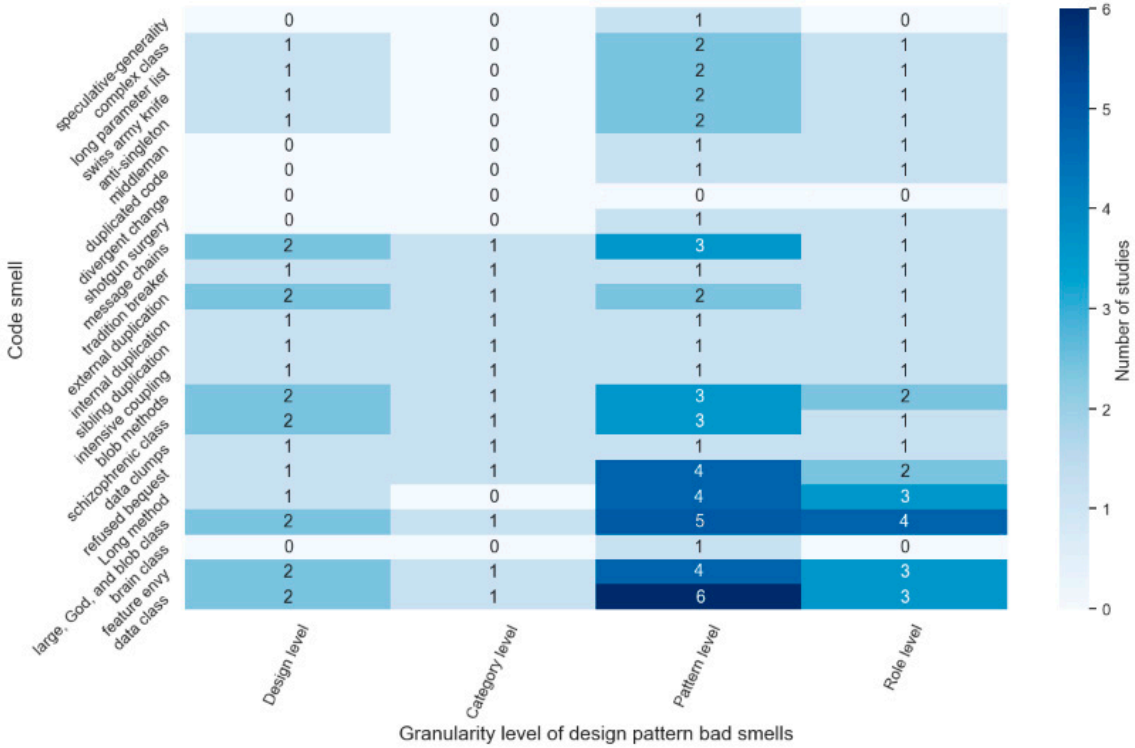

Figure 11. Distribution of code smell types over DBS granularity levels.

An interesting finding was also observed from the perspectives of grime occurrence in design patterns. All of the selected studies identified grime occurrences in a few DBGL of analysis, while focusing mainly on the pattern level. Thus, a heatmap is generated to show the relationship between the four types of grime and DBGL. Figure 12 shows a clear trend on the Modular grime occurrence at the pattern and role levels. Simultaneously, minor attention has been rendered to the design level and category level with an association to the Modular grime. Modular grime has a superiorly high occurrence in design patterns with many investigation studies following the Class grime. Behavioral grime and Organizational grime associated with the pattern level were limited to studies 1 and 2 only.

3.4. The Association of DBS to GoF Design Pattern Categories and Types

As previously mentioned, GoF design patterns' catalogue includes 23 patterns categorized into creational patterns, behavioral patterns, and structural patterns. Different GoF design patterns and categories were analyzed to understand the phenomenon of DBS occurrences. For example, the Factory method pattern and Prototype pattern from creational patterns were utilized to investigate the occurrences of DBS in their design structure [4,17,24]. However, amongst all the reviewed studies, we found that the DBS

(code smells and grime) occurrence research community primarily focused on analyzing creational patterns, particularly the Factory method pattern, Singleton pattern, and Prototype pattern. In contrast, Structural patterns received a lesser amount of attention in studies focused either on design pattern grime occurrence or code smell occurrence. Based on our findings, we have demonstrated various trends and discovered that all studies emphasized two or more pattern categories or combined different types of design patterns.

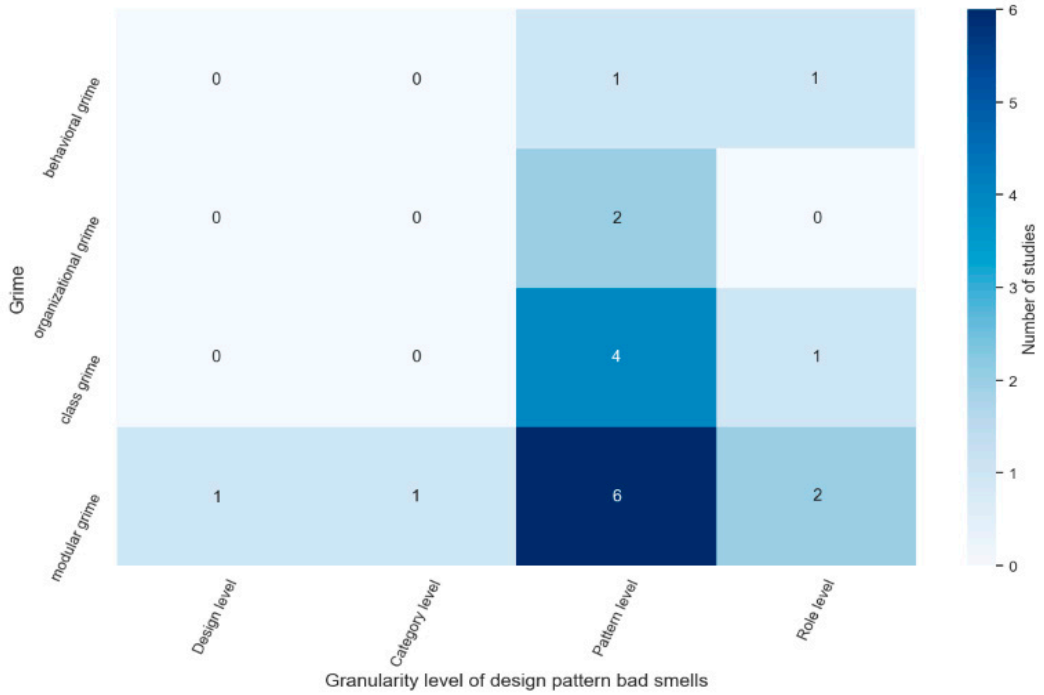

Figure 12. Distribution of grime types over DBS granularity level.

However, the studies which focused on investigating the occurrence of code smell types in design pattern showed various trends compared to the studies that focused on grime occurrence in design pattern instances. Concerning the code smell occurrence studies, seven out of nine studies investigated the Factory method pattern with code smell occurrences, such as Feature envy, Schizophrenic class, God and Blob classes. Whereas five studies focused on Prototype and Singleton with code smells such as Data class, External duplication, God and Blob classes, Refused bequest, and Blob method. In comparison, Decorator pattern, Composite pattern, and Adapter pattern are more frequently analyzed than structural patterns among studies 7, 6 and 5 with code smells, respectively. Therefore, behavioral patterns illustrate that Command, Observer, and Template method patterns are highly associated with code smell, such as Feature envy, Schizophrenic class, God and Blob classes, Data class, Long method, and Duplicated code. However, the Chain of Responsibility pattern was neglected throughout the studies (see Table 7).

On the other hand, Singleton and Factory method creational patterns were significantly investigated by 7 and 6 grime occurrence, such as Modular and Class grime. Furthermore, five out of seven studies aimed to examine the grime occurrence, such as Organizational, Behavioral, and Modular grimes, in both Adapter pattern and State pattern, from structural patterns and behavioral patterns. Therefore, Bridge pattern, Interpreter pattern, Memento pattern, and Chain of Responsibility pattern were completely ignored for investigating their association with grime occurrences (see Table 7).

3.5. DBS Detection Approaches and the Utilized Datasets

According to the DBS detection approaches, our analysis of the 16 studies found that the research community applied and proposed many different approaches and strategies to

detect DBS (grime and code smells) in design pattern structure and behavior. The utilized detection approaches could be categorized into five main approaches, summarized in Table 8.

Table 7. Summary of DBS occurrences in different types and categories of GoF design patterns.

DBS Types	Types of Pattern Categories	Design Pattern Types	Number of Studies	Paper ID
Code smell occurrences	Creational pattern	Abstract factory	1	1
		Builder	1	1
		Factory method	7	1, 2, 4, 6, 7, 8, 15
		Prototype	5	1, 2, 4, 7, 8,
		Singleton	5	1, 2, 7, 15, 16
	Structural pattern	Adapter	5	1, 2, 7, 15, 16
		Bridge	4	1, 2, 15, 16
		Composite	6	1, 2, 4, 7, 8, 15
		Decorator	7	1, 2, 4, 7, 8, 12, 15
		Façade	1	1
		Flyweight	1	12
		Proxy	4	1, 2, 7, 15
	Behavioral pattern	Command	7	1, 2, 4,7, 8, 9, 15
		Interpreter	1	12
		Iterator	1	1
		Mediator	2	1, 12
		Memento	2	1, 12
		Observer	6	1, 2, 4, 7, 8, 15
		State	5	1, 2, 7, 12, 15
		Strategy	5	1, 2,7, 12, 15
		Template method	6	1, 2, 7, 9, 15, 16
		Visitor	3	1, 2, 12
		Chain of Responsibility	0	
Grime occurrence	Creational pattern	Abstract factory	1	10
		Builder	0	
		Factory method	6	5, 6, 10, 11, 13, 14
		Prototype	3	5, 6, 10
		Singleton	7	3, 5, 6, 10, 11, 13, 14
	Structural pattern	Adapter	5	3, 5, 6, 10, 13
		Bridge	0	
		Composite	3	5, 6, 10
		Decorator	3	5, 6, 10,
		Façade	2	1, 10
		Flyweight	1	10
		Proxy	2	10, 13
	Behavioral pattern	Command	3	5, 6, 10
		Interpreter	0	
		Iterator	2	1, 13
		Mediator	2	1, 10
		Memento	0	
		Observer	4	5, 6, 10, 13
		State	5	3, 5, 6, 10, 13
		Strategy	3	5, 6, 10
		Template method	4	3, 5, 6,10
		Visitor	4	5, 6, 10, 13
		Chain of Responsibility	0	

We observed that five (31.25%) studies used the conformance checking-based approach for detecting grime occurrence in design pattern; they used design pattern Role-Based Meta-Modelling Language (RBML) to describe design pattern intent rules [44]. The procedure for checking the pattern instance conformance involves mapping the pattern members that exist in the implementation of the pattern at meta-level to its roles, which was captured through Structural-Pattern-Specification (SPS) and Interaction-Pattern-Specification (IPS), by using various algorithms such as the divide-and-conquer algorithm [1,2,19,26,27]. The

conformance checking approach showed its effectiveness to detect grime occurrence in the design pattern. However, this approach requires substantial effort to identify the RBML pattern rules, while these types of checks require upfront work on the developer or architect's part to define the rules of these semantic components prior to analyzing the structural integrity of the system. This method could be complex and may lead to false detection of grime occurrences if the rules are imprecisely identified [47].

Table 8. Summary of DBS detection approaches.

Approach	Type of Detected DBS	Number of Studies	Paper ID
Conformance Checking Approach	Grime	5	3, 10, 11, 13, 14
Metric-Based Approach	Grime/code smells	3	5, 6, 8
Machine Learning Algorithms	Code smells	1	16
Association Rule Mining Approach	Code smells	5	1, 2, 7, 9, 15
Rule-Based Approach	Code smells	2	4, 12

Similarly, another five studies out of the 16 mainly utilized association rule mining to detect code smells in the design pattern. The authors established association rules relying on three metrics: support, confidence, and conviction, using the following concepts: Transaction, which identifies the analyzed system classes; Antecedent, which indicates GoF design patterns investigated in the study; and Consequent, which indicates each explored code smell [4,18,36,38], with the help of algorithms such as Apriori algorithm for the detection purpose [17]. This method appeared to be helpful in discovering interesting relationships and correlations among the different items of the database. However, the association rule methods could suffer from obtaining non- interesting rules, or a huge number of discovered rules, and this could lead to low algorithm performance [48].

Furthermore, the metric-based and rule-based approaches were applied by three (18.75%) and two (12.50%) of the studies, respectively. They identified different metrics and rules for detecting grime or code smell occurrences in design pattern instances interchangeably. For instance, Freeman et al. [5] and Gamma [6] identified different metrics: number of alien attributes, number of alien public methods, and afferent pattern coupling for the detection of grime occurrences in design patterns (Modular grime, Class grime, Organizational grime). The advantage of using a rule-based approach and metric-based approach for DBS detection is that training data is not required. On the other hand, such approaches fall short when listing all possible rules for detections. Besides, the metric-based approach requires a huge calibration effort to find the best threshold values for each metric [49]. Interestingly, there is only one study that focused on utilizing machine learning algorithms to detect DBS; Kaur and Singh [39] used the J48 decision tree for detecting different code smell symptoms in pattern instances. The authors first formed the dataset for learning the J48 decision tree classifier for detection. They built the dataset by harmonizing the extracted classes by smell detection tools and the extracted classes by pattern detection tools. Finally, they used the J48 decision tree algorithm to perform the DBS detection. While the main benefit of the J48 decision tree is its simplicity in interpreting results, its decision depth affects the running complexity and utilizes a large storage space, as the values need to be stored in arrays frequently [50].

Additionally, we specified the domains focused by researchers for investigating and detecting DBS occurrences. The research community investigated DBS occurrence by extracting the utilized design patterns in various software systems. However, it manifested that the research of DBS occurrences usually focused on the software from the software modelling domain (10 studies) (e.g., ArgoUML is a software used for modelling UML diagrams, it uses design patterns in its design [1,4,23]), web application domain (eight studies), software programming domain (five studies), game applications domain (two studies), and a few other different types of domains.

Regarding the exploited datasets to conduct the empirical experiments, we observed that programs from Qualitas.class Corpus (i.e., ArgoUML, JfreeChart, PMD, JRefactory, JHotDraw, eclipse-SDK) were adopted by 11 (56.25%) studies for their empirical experiments. Therefore, we observed in most studies that Qualitas.class Corpus programs are adopted together with other subject programs (i.e., [2,18,23,35,37,39]). On the other hand, we found that there is only one study that employed a single dataset called P-Mart repository for conducting their empirical experiments (e.g., Kaur and Singh [39]). Therefore, our findings could conclude that the majority of studies used more than one program and dataset for conducting their empirical experiments [1,19,24,25].

4. Discussion

4.1. DBS Characteristics and Objectives

Referring to the first research question (RQ1), our findings indicate that DBS occurrence has been extensively studied recently with objectives linked to four DBGL. The pattern level promotion was strongly preferred as a primary objective for analyzing the occurrence of bad smells in design patterns, followed by the role level (e.g., [1,4,17,18,23,37,38]). The design level and category level were almost neglected for analyzing DBS occurrences. From the GoF design pattern categories' perspective, the central area of focus is analyzing behavioral patterns, followed by creational patterns and structural patterns.

As a critical finding. investigating DBS occurrence in category level, design level, and role level is not sufficiently supported by the DBS research community. Thus, only two studies were supported by analyzing DBS occurrence in GoF pattern categories at the Category level. Therefore, based on several studies, we know how significant it is to promote the Category level in relation to the three GoF categories (creational patterns, structural patterns, and behavioral patterns). Knowing the co-occurrence degree of DBS with each GoF pattern category would highly increase the developers' appreciation of this phenomenon and, in turn, the developers would be more precise while applying patterns from that GoF category in their software design [17,40].

Similarly, DBS at the design level helps developers and researchers comprehensively measure the degree of accumulating bad smells in GoF design pattern classes to identify DBS's effect on various software quality attributes and maintenance costs [17,25]. Furthermore, research of DBS occurrence should focus significantly more on the role level because it is a key to accurately understand and detect the occurrence of bad smells in design patterns as many bad smells begin to occur from the wrong extension and implementation of pattern's roles [37].

Initially, GoF design patterns included 23 pattern types that were categorized under structural patterns, creational patterns, and behavioral patterns [6]. The most targeted design pattern types associated with pattern level of analysis are Decorator pattern, Adapter pattern, and Composite pattern from the structural patterns category; Singleton, Prototype, and Factory method patterns from the creational patterns category; and Observer, Template method, State, and Commend patterns from the behavioral patterns category. One possible explanation is that the GoF design patterns detection tools are more readily available to detect those pattern types and be used more frequently with good precision and recall [4,17,51,52].

4.2. Types of DBS Occurrences and Their Association with DBS Granularity Levels and Design Pattern Types and Categories: A Sustainability Perspective

In addressing the second research question (RQ2), we observed two main categories of bad smells that occur with GoF design patterns: code smells and grime. We identified 28 DBS, out of which 24 were under code smells and four were under the grime occurrence category. We found that the Data class, God class, Feature envy, Large class, Long method, Blob class, and Refused bequest code smells symptoms widely co-occur with design patterns. On the other hand, Modular grime is the most explored grime. indicating high occurrences with design patterns. These smells were found to be a threat to the sustainabil-

ity and evolution of software; however, their occurrences are explainable as the emergence of the code smells symptoms are extensively contributed by software developers unintentionally. For example, Sousa et al. [4] argued that the Proxy pattern presented a high level of occurrences with Data class smell because the developer may identify many attributes and assign many methods, such as gets and sets, to access the stored value in Proxy classes, which in turn, contribute to the accumulation of Data class symptoms. Besides, developers tend to extend design patterns during software evolution by adding relationships with either a new pattern or non-pattern classes. Therefore, during the extension, developers may violate the principles of design pattern, where any non-awareness extension of pattern relationships will significantly raise the coupling of the classes and entail modular grime occurrence, which destroys the design patterns' integrity [2,25,53].

Code smells and grime occurrences were explored with creational patterns, structural patterns, and behavioral patterns. Reliance on structural patterns are less favored, compared to creational patterns and behavioral patterns in grime occurrence, because structural patterns are more related to the system structure which might be less prone to change [54,55]. Besides, the main intent of the structural pattern is the ease of grasping them, as applicable in software design [8]. More specifically, code smells and grime studies have been associated with different GoF design pattern types; therefore, they rely heavily on Factory method, Singleton, Adapter, and Template method patterns. As they are the most utilized design pattern types amongst software developers, and their structure and intent are more change-prone during software evolution, they tend to accumulate more bad smells [4,17,24,25].

Analyzing DBS occurrences at different granularity levels is critical to understanding the degree of DBS accumulation and their impact on software sustainability and quality. Thus, we observed the relationship between DBS and DBGL that showed a great promotion of pattern level associated with Data class, Feature envy, God, Blob, and Large classes, Long method, and Refused bequest. Class grime and Modular grime are highly associated with the pattern level of analyses. Role level was widely investigated in code smell symptoms than grime symptoms. However, the category and design level received less attention through the selected studies. We witnessed that Modular grime is the only grime symptom analyzed at the design and category level. In addition, category and design levels were investigated less with code smells compared to the role and pattern levels. We arrived at this conclusion because the pattern level is the most significant level of analysis, as each DBS symptom would be associated with at least one class within one pattern. DBS occurrences at the pattern level indicate a direct impact on the entire structure of the pattern [1]. However, that does not mean design, category, and role levels are less significant, as each DBS symptom may have different impacts at each granularity level, which is a threat to the sustainability and evolution of software at different degrees and dimensions.

Thus, DBS occurrences were found to affect two dimensions of sustainability: the technical dimension and the environmental dimension. To preserve the sustainability of software, researchers need to analyze DBS at different DBGL levels. From a software sustainability perspective, the following subsection discusses the importance of analyzing DBS occurrences at different levels of DBGL.

4.2.1. Technical Dimension

The technical dimension focuses on the longevity of software systems (e.g., maintainability, evolution) [56–58]. Several studies found that DBS occurrence affects a variety of the software maintenance aspects. For instance, the occurrence of behavioral grime significantly influences the stability of the software. Stability is one of the vital software sustainability characteristics which affects software evolution and maintainability [59,60]. The occurrence of Modular grime and Class grime impacts software understandability, testability, and usability [19,24]. Hence, Complex and God classes increase the complexity of the code and affect the code testability [17]. Furthermore, classes with God and Brain class smells exhibited a negative impact on software change size, frequency, and

defects [61]. Additionally, the Blob class affects program understandability and maintainability [17]. Numerous studies showed that the occurrence of grime introduces a technical debt [26,46]. For instance, Dale and Izurieta [26] dedicated a Modular grime to having a potential negative impact on technical debt and maintainability aspects.

Based on the above observations, DBS occurrences affect the technical dimension of software sustainability by impacting the various software maintenance characteristics. However, analyzing DBS occurrences at different granularity levels would assist in preserving the technical dimension of sustainability. For example, Modular grime, Feature envy, and Data class smells begun to be initiated from the wrong implementation of pattern roles [1,2,17]. Thus, exploring the DBS occurrence at the role level would help developers understand and manage the situation, where some smells can initiate and assist developers to eliminate DBS adverse effects by monitoring the pattern's roles and performing refactoring when needed. Furthermore, DBS at the design level plays a critical role in its impact on software maintainability by increasing the software maintenance cost. It also helps the developers control DBS growth, estimate their severity on maintenance cost, and perform refactoring where necessary in order to balance the technical dimension of sustainability.

4.2.2. Environmental Dimension

The green or environmental dimension of software sustainability aims to develop a sustainable software product by promoting energy efficiency and reducing the environmental impact on its support [62]. However, DBS occurrence (code smells and grime) threatens the green dimension of software sustainability by increasing the energy consumption of software applications. In other words, DBS increases the energy leak by reducing the software's performance with the execution of useless code [22,24]. For example, Reimann et al. [63] examined the code smells impact on energy consumption. They observed that Duplicate code, God Class, Long Method, and Feature envy code smells tend to increase the code size and undesirably change code execution.

Moreover, performance is one characteristic that correlates with energy consumption as several authors reported that software with higher performances also have higher energy efficiencies [64]. Numerous studies showed the impact of grime occurrence on software performance [24,44]. For instance, behavioral grime occurrence in design patterns harms the runtime behavior of the pattern, which negatively affects software performances [1].

To measure the impact of DBS on the energy consumption of software applications and preserve the environmental dimension of software sustainability, researchers need to analyze DBS occurrences at different granularity levels, such as the design level and category level. As we had mentioned before, the design level estimates the growth of different DBS occurrences in design pattern classes which are employed in software design. Thus, analyzing DBS occurrence at design levels is very critical, as it assists software developers in estimating the degree of impact of DBS on software energy consumption and performance. Besides that, Reimanis and Izurieta [1] showed that behavioral grime is more correlated with the behavioral pattern category. Thus, analyzing DBS occurrence at the category level is very vital in maintaining software sustainability. It can assist the developers to determine which design pattern category is more prone to accumulate DBS and affects the software's energy consumption and performance.

4.3. DBS Detection and the Utilized Datasets

With respect to the third research question (RQ3)'s finding, we recognized that several sophisticated detection approaches exist for DBS occurrence. However, the area is oriented more towards using a conformance checking-based approach for detecting grime occurrence in design patterns and an association rule mining approach for detecting code smell occurrences in design patterns [3,5,6,10,11,13,14]. This distribution is partially due to the fact that detecting grime occurrence in design patterns is highly dependent on capturing the deviation of the original pattern intent. Therefore, the conformance-checking approach facilitates the mapping between the applied design pattern and original pattern

intent for checking the existence of any grime symptoms. However, this approach requires substantial effort from the developer to identify the RBML pattern rules of these semantic components prior to analysing the pattern's structural and behavioral integrity [47].

The association rule mining approach allows for the combining of dataset items using different metrics such as support, confidence, lift, and conviction to mine knowledge about the data; thus, it facilitates the detection of code smell occurrences in design patterns. Therefore, the association rule mining approach could suffer from generating non-interesting rules, or a huge number of rules [48]. Additionally, we noticed that there is a particularly limited number of studies using machine learning algorithms for grime occurrence detection. However, machine learning algorithms showed a great achievement for many detection purposes such as detecting defects in source codes [65]. Thus, more attention should be given to develop machine learning algorithms for detecting DBS occurrences. From the perspective of the utilized datasets, we observed that Qualitas Class Corpus is the most utilized dataset for the empirical experiments. This is because many authors are encouraged to utilize Qualitas class in their empirical experiments as it involves different large programs with a good number of integrating GoF design patterns in their system design [4,36,38,39]. When summarizing our findings we could identify a number of challenges faced by the DBS research domain.

Refactoring DBS challenges: the occurrence of DBS (code smells and grime) has provided harmful impacts on both design pattern and software quality. They increase source code complexity, resulting in the decline of software understandability, correctness, and the increase in the fault and defect-proneness of software, proving that their exhiliration degrades the patterns' integrity, reusability, and adaptability [24,26,28]. However, an under-explored challenge is refactoring the design pattern's code fragments, which is affected by the occurrence of DBS. Identifying a refactoring mechanism by restructuring the design pattern body of code containing bad smells and altering its internal structure without changing their external behavioral helps to eliminate the growth of DBS and their negative impact [27]. The removal of DBS as they appear can potentially control the maintenance cost and improve the adaptability and test effectiveness, and transform a misdesign and bad code into clean code and simple design [2].

Factors of DBS emergence challenges: currently, the research community identified DBS types which co-occur with GoF design patterns and examined their influence on a variety of DBGL. However, the current research has limited depth with regards to the factors which could contribute to the emergence of DBS occurrences. For example, developers with different levels of background, experiences, and project involvement may produce different trends for accumulating DBS in GoF pattern structures [3]. Additionally, projects under development may involve various characteristics in terms of application domain, types of systems (such as user applications, which would affect the usage of GoF patterns), and the development practices as different projects can accumulate DBS differently. Thus, emphasizing the factors related to the high levels of DBS occurrences can enhance the awareness of the usage and the impact of GoF design patterns over the quality of software.

GoF DBS detectors challenge: numerous DBS detection approaches have been proposed and utilized. A majority of the approaches need to firstly extract design pattern classes from the system by using the help of GoF design pattern detection tools and use them as inputs to the DBS tools for detection purposes [24,38]. That means all the proposed approaches are not capable of recognizing GoF design pattern instances in their integrated form with software codes and categorizing them for the prioritization of their refactoring over the other code fragments. Consequently, DBS detection tools need to pay more attention to such issues and be aware of GoF design patterns' instances and structures as this increases the effectiveness of the tools in detection DBS that could result in more maintenance efforts. Besides that, most of the proposed approaches relied on conformance checking for detecting grime occurrences in design patterns, while there is a limit for applying machine learning algorithms for grime occurrence detection. Therefore, machine learning algorithms showed a great potential for the purpose of detection within the different fields of research (e.g.,

using support vector machine (SVM) for detecting logistic packing defects [65]). Thus, the research community needs to place more attention on benefiting from machine learning technology in DBS occurrence detection, particularly grime occurrences.

Granularity challenge: The research domain examined the occurrence of DBS according to four DBGL: design level, category level, pattern level, and role level, to check if the nature of GoF design pattern varies in their granularity and levels. Furthermore, GoF design patterns were applied to two levels, primarily to classes or to objects; this is classified as the scope level of GoF design patterns [6]. Class patterns mainly concentrate on managing the relationships between classes and their subclasses. In contrast, the inheritance relationship establishes the static and fixed relationships at compile time. The objects' patterns manage the relationships between objects rather than classes, which are changeable at run time and are more dynamic [6]. Thus, the association of scope levels of GoF design patterns and DBS are unexplored and unexamined, whereas exploring the occurrence of DBS in scope levels of GoF design patterns will significantly strengthen the developers and researchers' awareness of the primary keys for causing DBS occurrence in GoF design patterns.

DBS severity challenge: generally, Ampatzoglou et al. [60] and Khomh et al. [54] stated that GoF design pattern classes can cover between 15% and 65% of the system design. This fact clarifies that GoF design patterns significantly affect the quality of the system. Hence, the degree of accumulation DBS in design pattern classes would damage the entire software quality [24]. Based on our analysis of the existing body of knowledge, we noticed that a majority of the studies focused on identifying DBS co-occurrence with patterns' class and totally neglected the importance of identifying the degree of severity of accumulating DBS pattern instances. Consequently, the severity information of the occurrence of DBS in GoF design patterns would significantly assist the developers to prioritize the refactoring tasks. Considering the DBS severity relationship with GoF design patterns, developers will be able to prioritize different parts of design pattern classes and code. For example, if design patterns' class was affected by more than one DBS type, developers would need to provide prioritizations for refactoring patterns' class in order to prevent damaging the entire patterns' structure and quality.

5. Limitations

Our study was restricted by our eligibility criteria; specifically by limiting our SLR to the studies which address DBS. If our study had not been regulated in this way, we would have worked on a wider variety of references. Furthermore, even within the DBS occurrence field, we eliminated many DBS studies which discussed the occurrence of bad smells such as anti-patterns, code smells, and grimes, which showed no robust study design and results. In addition, studies which investigated the consequences of applying GoF design patterns on software quality (e.g., change-proneness, defect, and fault density) were further excluded. Unfortunately, the authors of these studies had not thoroughly emphasized the occurrence of DBS in design patterns' instance in detail, preventing us from conducting a meta-analysis to investigate the consequences of DBS occurrences on design patterns and software quality.

6. Conclusions

In this SLR, we reviewed 16 studies which met the identified eligibility criteria to answer the SLR research questions, focusing on DBS occurrences in GoF design patterns. We found that the research community investigated DBS occurrence in four DBGL levels, which are the design level, category level, pattern level and role level. Indeed, the research field strongly investigated DBS occurrences in pattern level and role level as primary objectives of their studies. GoF design patterns categorise behavioral patterns, structural patterns, and creational patterns which were investigated interchangeably in all studies. The behavioral patterns appear to be the most targeted group of patterns for investigating DBS occurrence phenomenon in promoting DBS granularity levels (DBGL). Category levels

were rarely investigated, pattern level and role levels were the most supported, and design levels were not investigated that often by DBS.

With regards to DBS types, we identified two main types of DBS's that more frequently co-occur with GoF design patterns: code smell occurrence and grime occurrence. Additionally, we identified four types of grime investigated in the studies which focused on the design pattern grime occurrence phenomenon, and 24 code smells associated with GoF design patterns analyzed in the studies which concentrated on design pattern code smell occurrences. The investigation of GoF pattern occurrence with DBS shows that they vary in their types and categories. In contrast, the selected studies primary focused on analyzing DBS over pattern level, role level, or a combination with either the category level, design level, or both.

Code smells and grime occurrence analysis are geared more towards the pattern level, or are combined with role, design, and category levels. Concerning pattern types, the Factory method, Singleton, Adapter and State patterns are preferred for investigating code smells and grime studies, as they have a higher level of association with code smell occurrences. Two main code smell types, reliant on the data class, and Feature envy associated with the pattern level of analysis, are more targeted for analyzing code smell occurrences in GoF design patterns. Furthermore, grime types are mostly associated with Modular grime and Class grime in relation to the pattern level. Thus, code smells and grime appear to be crucial for analyzing the symptoms of bad smells and design patterns' degeneration.

From the sustainability aspect, we found that two dimensions of sustainability are affected by DBS, the technical dimension and the environmental dimension. DBS greatly impacts software understandability, testability, usability, code testability, change in size, frequency, and defects, further affecting program understandability and maintainability. This result is a direct threat to the technical dimensions of software sustainability. Besides, the occurrences of DBS in design patterns could increase the energy consumption of software applications which, in turn, affects the stability of the green dimension of software sustainability.

Additionally, we also highlighted the utilized approaches for DBS detection. The conformance checking approach is the most preferred one for detecting grime occurrences. Furthermore, association rule mining is among the most used approaches to detect code smells in GoF design patterns. Our findings demonstrate that, regarding the used dataset for conducting empirical experiments, Qualitas.class Corpus is a widely adopted dataset for empirically investigating DBS. However, some studies used Qualitas.class Corpus with different individual software programs. The P-Mart repository was established mainly for design patterns, empirical investigations, and studies, although it was the least used dataset amongst all the selected studies.

Based on the findings of this study, we identified five main future directions and challenges. First, there is an under-explored future direction towards refactoring the code fragment of DBS to remove DBS as they appear, and eliminate their negative impact on software quality and sustainability. Also, there is a need for investigating the factors which contribute to the emergence of DBS to improve the research community awareness about the usage and impact of GoF design patterns over quality. Moreover, we found that further research needs to be explored for detecting DBS by recognizing GoF design patterns in their integrated form in software codes, as most of the proposed approaches need the help of design pattern detection tools to recognize the existing pattern instance to detect the DBS.

Furthermore, the research community examined the occurrences of DBS according to four DBGL, whether the nature of GoF design pattern varies in their granularity and levels. Thus, exploring the occurrence of DBS regarding GoF granularity scope levels will significantly assist in discovering the primary keys to initiate DBS occurrences in the GoF design patterns. Consequently, measuring and identifying the degree of severity of DBS is very critical and needs further research. This identification will help the developers prioritize

the refactoring tasks, and further prioritize the different parts of design pattern classes and code that need refactoring before violating the entire pattern's structure and quality.

Supplementary Materials: The identified bad smells' occurrences in design patterns are available online at https://www.dropbox.com/s/7i1zt636m942zxe/Bad%20smells%20occurrence%20in%20GoF%20design%20patterns%20%20%281%29.xlsx?dl=0 (accessed on 1 September 2021).

Author Contributions: S.H.S.A.: Conceptualization, Data curation, Formal analysis, Methodology, Visualization, Writing—review & editing. D.H.: Conceptualization, Data curation, Formal analysis, Methodology, Visualization, Supervision, Writing—review & editing. R.B.A.: Conceptualization, Methodology, Supervision, Funding acquisition, Writing—review & editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Fundamental Research Grant Scheme (FRGS/FP060-2020).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: This is a literature review, all the articles are cited.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

In this section, we identified 28 types of DBS under two main categories (code smells and grime). Code smells include 24 types that co-occur with GoF design patterns, while grime involves 4 main types that occur with design patterns.

Table A1. The description of grime occurrence types in design patterns.

DBS Category	Type		Description
Grime	Behavioral grime		Behavioral grime refers to the deviations observed from a flow of information perspective that captures the operational side of a design pattern at run time which could be reflected by UML sequence diagrams. It indicates two symptoms: (a) improper order of sequences: in which expected behaviors occur in an incorrect order; (b) excessive actions, in which excessive actions hamper the run-time expectations of a pattern.
	Structural grime	Modular grime	Modular grime is a symptom of build-up unrelated relationships-generalizations, realizations, associations, dependencies-to pattern responsibilities.
		Class grime	Class grime indicates increasing the number of methods and public attributes in pattern classes that are not relevant to the pattern's responsibilities.
		Organizational grime	Organizational grime is a symptom of increasing coupling of the pattern files and namespaces that are not relevant to pattern responsibilities.

Table A2. The description of code smell occurrence types in design patterns.

DBS Category	Type		Description
Code Smells	Complex class		Indicates that the class is too complex, includes several complex methods, and is very difficult to understand.
	Data class		Indicates a symptom of class that only holds fields and crude methods for accessing them.
	Feature envy		Indicates a symptom of methods that accesses data of another object more than its own data.
	Brain Class		Indicates a symptom of class that is complex and centralizes the functionality of the system.
	Long method		Indicates a method or function that has grown too large in terms of LOCs.
	Refused bequest		Indicates a class inherited from a base class; however, not all the inherited behavior are needed.
	Large class/God Class/blob class		Indicates classes which operate most of the work, have too many responsibilities, and excessively large and complex.
	Data Clumps		Indicates that different parts of the codes, including similar groups of variables should be transformed into their own classes.
	Schizophrenic Class		Indicates a class that captures two or more key abstractions.
	Blob Methods		Indicates a class that holds most of the processing and executes most of the decisions.
	Duplication	Sibling Duplication	Indicates a duplication between siblings in an inheritance hierarchy.
		Internal Duplication	Indicates methods that are related to the same class or module.
		External Duplication	Indicates unrelated operations.
		Duplicated Code	Indicates a very similar code that is repeated at different locations.
	Intensive Coupling		Indicates a class that calls many other methods from a few classes.
	Tradition Breaker		Indicates classes that do not use their parents protected members.
	Message Chains		Indicates a long list of call methods.
	Shotgun surgery		Indicates a method called by many classes, many times.
	Divergent Change		Indicates classes with many changes made to them.

Table A2. *Cont.*

DBS Category	Type	Description
	Middelman	Indicates a class that implements one action, while assigning the work to another class.
	AntiSingleton	Indicates a class that has changeable variables, which could be used as global variables.
	SwissArmykinfe	Indicates a class that has a set of many methods, providing unrelated functionalities.
	Long Parameter List	Indicates a class that has at least one method with a long list of parameters.
	Speculative Generality	Indicates a class that is an abstract class; however, it has few children that do not use its methods.

References

1. 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 Science and Business Media LLC: Berlin/Heidelberg, Germany, 2019; pp. 77–93.
2. Izurieta, C.; Bieman, J.M. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Softw. Qual. J.* **2013**, *21*, 289–323. [CrossRef]
3. Feitosa, D.; Ampatzoglou, A.; Avgeriou, P.; Chatzigeorgiou, A.; Nakagawa, E.Y. What can violations of good practices tell about the relationship between GoF patterns and run-time quality attributes? *Inf. Softw. Technol.* **2019**, *105*, 1–16. [CrossRef]
4. Sousa, B.L.; Bigonha, M.; Ferreira, K.A.M. An exploratory study on cooccurrence of design patterns and bad smells using software metrics. *Softw. Pract. Exp.* **2019**, *49*, 1079–1113. [CrossRef]
5. Freeman, E.; Robson, E.; Bates, B.; Sierra, K. *Head First Design Patterns*; O'Reilly Media, Inc: Newton, MA, USA, 2004.
6. Gamma, E. *Design Patterns: Elements of Reusable Object-Oriented Software*; Pearson Education India: Delhi, India, 1995.
7. Voorhees, D.P. Introduction to Design Patterns. In *The Data Science Design Manual*; Springer Science and Business Media LLC: Cham, Switzerland, 2020; pp. 389–402.
8. Elish, M.O.; Mohammed, M.A. Quantitative analysis of fault density in design patterns: An empirical study. *Inf. Softw. Technol.* **2015**, *66*, 58–72. [CrossRef]
9. Onarcan, M.O.; Fu, Y. A Case Study on Design Patterns and Software Defects in Open Source Software. *J. Softw. Eng. Appl.* **2018**, *11*, 249–273. [CrossRef]
10. Riehle, D. Lessons Learned from Using Design Patterns in Industry Projects. In *Transactions on Pattern Languages of Programming II*; Springer Science and Business Media LLC: Berlin/Heidelberg, Germany, 2011; Volume 6510, pp. 1–15.
11. Beck, K.; Crocker, R.; Meszaros, G.; Coplien, J.O.; Dominick, L.; Paulisch, F.; Vlissides, J. Industrial experience with design patterns. In Proceedings of the IEEE 18th International Conference on Software Engineering, Montreal, QC, Canada, 1–6 October 2002; pp. 103–114.
12. Vokáč, M.; Tichy, W.; Sjøberg, D.I.K.; Arisholm, E.; Aldrin, M. A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment. *Empir. Softw. Eng.* **2004**, *9*, 149–195. [CrossRef]
13. Zafeiris, V.E.; Poulias, S.H.; Diamantidis, N.; Giakoumakis, E. Automated refactoring of super-class method invocations to the Template Method design pattern. *Inf. Softw. Technol.* **2016**, *82*, 19–35. [CrossRef]
14. Turkistani, B.; Liu, Y. Reducing the Large Class Code Smell by Applying Design Patterns. In Proceedings of the 2019 IEEE International Conference on Electro Information Technology (EIT), Brookings, SD, USA, 20–22 May 2019; Institute of Electrical and Electronics Engineers (IEEE): Piscataway, NJ, USA, 2019; pp. 590–595.
15. Ouni, A.; Kessentini, M.; Cinnéide, M.Ó.; Sahraoui, H.; Deb, K.; Inoue, K. MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells. *J. Softw. Evol. Process* **2017**, *29*, e1843. [CrossRef]
16. Arcelli, D.; Di Pompeo, D. Applying Design Patterns to Remove Software Performance Antipatterns: A Preliminary Approach. *Procedia Comput. Sci.* **2017**, *109*, 521–528. [CrossRef]
17. Alfadel, M.; AlJasser, K.; Alshayeb, M. Empirical study of the relationship between design patterns and code smells. *PLoS ONE* **2020**, *15*, e0231731. [CrossRef] [PubMed]
18. Walter, B.; Alkhaeir, T. The relationship between design patterns and code smells: An exploratory study. *Inf. Softw. Technol.* **2016**, *74*, 127–142. [CrossRef]
19. 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, Torino, Italy, 18–19 September 2014; pp. 1–4.
20. Sousa, B.; Bigonha, M.; Ferreira, K.A.M. A systematic literature mapping on the relationship between design patterns and bad smells. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, ACM, Pau, France, 9–13 April 2018; pp. 1528–1535.
21. Mayvan, B.B.; Rasoolzadegan, A.; Yazdi, Z.G. The state of the art on design patterns: A systematic mapping of the literature. *J. Syst. Softw.* **2017**, *125*, 93–118. [CrossRef]
22. Sehgal, R.; Mehrotra, D.; Nagpal, R.; Sharma, R. Green software: Refactoring approach. *J. King Saud Univ. Comput. Inf. Sci.* **2020**. [CrossRef]
23. Hussain, S.; Keung, J.; Sohail, M.K.; Khan, A.A.; Ahmad, G.; Mufti, M.R.; Khatak, H.A. Methodology for the quantification of the effect of patterns and anti-patterns association on the software quality. *IET Softw.* **2019**, *13*, 414–422. [CrossRef]

24. Feitosa, D.; Ampatzoglou, A.; Avgeriou, P.; Nakagawa, E.Y. Correlating Pattern Grime and Quality Attributes. *IEEE Access* **2018**, *6*, 23065–23078. [CrossRef]
25. Feitosa, D.; Avgeriou, P.; Ampatzoglou, A.; Nakagawa, E.Y. The Evolution of Design Pattern Grime: An Industrial Case Study. In *International Conference on Product-Focused Software Process Improvement, Innsbruck, Austria, 29 November–1 December 2017*; Springer Science and Business Media LLC: Berlin/Heidelberg, Germany, 2017; pp. 165–181.
26. Dale, M.; 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—ESEM '14, Torino, Italy, 18–19 September 2014; pp. 1–4. [CrossRef]
27. 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, Bolzano-Bolzen, Italy, 16–17 September 2010; pp. 1–8.
28. Alkhaeir, T.; Walter, B. The Effect of Code Smells on the Relationship Between Design Patterns and Defects. *IEEE Access* **2021**, *9*, 3360–3373. [CrossRef]
29. Imran, A.; Kosar, T. Software Sustainability: A Systematic Literature Review and Comprehensive Analysis. *arXiv* **2019**, arXiv:1910.06109.
30. Kitchenham, B.; Brereton, O.P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering—A systematic literature review. *Inf. Softw. Technol.* **2009**, *51*, 7–15. [CrossRef]
31. Kitchenham, B.; Charters, S. Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3, EBSE Technical Report EBSE-2007-01, Keele University and University of Durham. 2007. Available online: https://www.bibsonomy.org/bibtex/aed0229656ada843d3e3f24e5e5c9eb9 (accessed on 20 August 2021).
32. Webster, J.; Watson, R.T. Analyzing the past to prepare for the future: Writing a literature review. *MIS Q.* **2002**, *26*, xiii–xxiii.
33. Kitchenham, B.; Pretorius, R.; Budgen, D.; Brereton, O.P.; Turner, M.; Niazi, M.; Linkman, S. Systematic literature reviews in software engineering—A tertiary study. *Inf. Softw. Technol.* **2010**, *52*, 792–805. [CrossRef]
34. Nidhra, S.; Yanamadala, M.; Afzal, W.; Torkar, R. Knowledge transfer challenges and mitigation strategies in global software development—A systematic literature review and industrial validation. *Int. J. Inf. Manag.* **2013**, *33*, 333–355. [CrossRef]
35. Jaafar, F.; Guéhéneuc, Y.-G.; Hamel, S.; Khomh, F.; Zulkernine, M. Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. *Empir. Softw. Eng.* **2016**, *21*, 896–931. [CrossRef]
36. Cardoso, B.; Figueiredo, E. Co-Occurrence of Design Patterns and Bad Smells in Software Systems: An Exploratory Study. In *Anais do XI Simpósio Brasileiro de Sistemas de Informação (SBSI)*; Sociedade Brasileira de Computação: Porto Alegre, Brazil, 2015.
37. Speicher, D. Code Quality Cultivation. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*; Springer Science and Business Media LLC: Berlin/Heidelberg, Germany, 2013; pp. 334–349.
38. Sousa, B.; Bigonha, M.; Ferreira, K. Evaluating Co-Occurrence of GOF Design Patterns with God Class and Long Method Bad Smells. In *Anais do Simpósio Brasileiro de Sistemas de Informação (SBSI)*; Sociedade Brasileira de Computação: Agronomia, Brazil, 2017; pp. 396–403.
39. Kaur, A.; Singh, S. Detecting software bad smells from software design patterns using machine learning algorithms. *Int. J. Appl. Eng. Res.* **2018**, *13*, 10005–10010.
40. Mohammed, M.A.; Elish, M.O. Empirical assessment of design patterns' fault-proneness at different granularity levels. *Adv. Comput. Des.* **2017**, *2*, 293–311.
41. Fowler, M.; Beck, K.; Brant, J.; Opdyke, W. *Refactoring: Improving the Design of Existing Code*; Addison-Wesley: Boston, MA, USA, 1999.
42. Lanza, M.; Marinescu, R. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, And Improve The Design Of Object-Oriented Systems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2007.
43. Brown, W.J.; Malveau, R.C.; McCormick III, H.W.; Mowbray, T.J. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1998.
44. Izurieta, C.; Reimanis, D.; Griffith, I.; Schanz, T. Structural and Behavioral Taxonomies of Design Pattern Grime. In Proceedings of the 12th Seminar on Advanced Techniques & Tools for Software Evolution, SATToSE, Bolzano, Italy, 8–10 July 2019; pp. 8–10.
45. Griffith, I.; Izurieta, C. Design pattern decay: An extended taxonomy and empirical study of grime and its impact on design pattern evolution. In Proceedings of the 11th ACM/IEEE International Doctoral Symposium on Empirical Software Engineering and Measurements, Baltimore, MD, USA, 10–11 October 2013.
46. Reimanis, D.; Izurieta, C. Towards Assessing the Technical Debt of Undesired Software Behaviors in Design Patterns. In Proceedings of the 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD), Raleigh, NC, USA, 4 October 2016; pp. 24–27. [CrossRef]
47. Whiting, E.; Andrews, S. Drift and Erosion in Software Architecture: Summary and Prevention Strategies. In Proceedings of the 2020 the 4th International Conference on Information System and Data Mining, Hilo, HI, USA, 15–17 May 2020; pp. 132–138.
48. Moreno, M.N.; Segrera, S.; López, V.F. Association Rules: Problems, solutions and new applications. In *Actas del III Taller Nacional de Minería de Datos y Aprendizaje*; 2005; pp. 317–323. Available online: http://www.lsi.us.es/redmidas/CEDI/papers/892.pdf (accessed on 1 September 2021).
49. dos Reis, J.P.; e Abreu, F.B.; de Figueiredo Carneiro, G.; Anslow, C. Code Smells Detection and Visualization: A Systematic Literature Review. In *Archives of Computational Methods in Engineering*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 1–48.
50. Mathuria, M. Decision tree analysis on j48 algorithm for data mining. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2013**, *3*, 1114–1119.

51. Zanoni, M.; Fontana, F.A.; Stella, F. On applying machine learning techniques for design pattern detection. *J. Syst. Softw.* **2015**, *103*, 102–117. [CrossRef]
52. Yu, D.; Zhang, P.; Yang, J.; Chen, Z.; Liu, C.; Chen, J. Efficiently detecting structural design pattern instances based on ordered sequences. *J. Syst. Softw.* **2018**, *142*, 35–56. [CrossRef]
53. Izurieta, C.; Bieman, J.M. Testing Consequences of Grime Buildup in Object Oriented Design Patterns. In Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation, Lillehammer, Norway, 9–11 April 2008; Institute of Electrical and Electronics Engineers (IEEE): Los Alamitos, CA, USA, 2008; pp. 171–179.
54. Khomh, F.; Gueheneuc, Y.-G.; Antoniol, G. Playing roles in design patterns: An empirical descriptive and analytic study. In Proceedings of the 2009 IEEE International Conference on Software Maintenance, Edmonton, UK, 20–26 September 2009; Institute of Electrical and Electronics Engineers (IEEE): Los Alamitos, CA, USA, 2009; pp. 83–92.
55. Posnett, D.; Bird, C.; Dévanbu, P. An empirical study on the influence of pattern roles on change-proneness. *Empir. Softw. Eng.* **2010**, *16*, 396–423. [CrossRef]
56. García-Berna, J.A.; De Gea, J.M.C.; Ros, J.N.; Fernández-Alemán, J.L.; Nicolás, J.; Toval, A. Surveying the Environmental and Technical Dimensions of Sustainability in Software Development Companies. *Appl. Sci.* **2018**, *8*, 2312. [CrossRef]
57. Le, D.M.; Carrillo, C.; Capilla, R.; Medvidovic, N. Relating Architectural Decay and Sustainability of Software Systems. In Proceedings of the 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, Italy, 5–8 April 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 178–181.
58. Skalka, J.; Drlik, M.; Benko, L.; Kapusta, J.; Rodríguez del Pino, J.C.; Smyrnova-Trybulska, E.; Stolinska, A.; Svec, P.; Turcinek, P. Conceptual Framework for Programming Skills Development Based on Microlearning and Automated Source Code Evaluation in Virtual Learning Environment. *Sustainability* **2021**, *13*, 3293. [CrossRef]
59. Ahmad, R.; Hussain, A.; Baharom, F. Software sustainability characteristic for software development towards long living software. *Environment* **2018**, *20*, 34.
60. Ampatzoglou, A.; Chatzigeorgiou, A.; Charalampidou, S.; Avgeriou, P. The Effect of GoF Design Patterns on Stability: A Case Study. *IEEE Trans. Softw. Eng.* **2015**, *41*, 781–802. [CrossRef]
61. Olbrich, S.M.; Cruzes, D.S.; Sjoberg, D.I. Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In Proceedings of the 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 1–10.
62. García-Mireles, G.A.; Moraga, M.Á.; García, F.; Calero, C.; Piattini, M. Interactions between environmental sustainability goals and software product quality: A mapping study. *Inf. Softw. Technol.* **2018**, *95*, 108–129. [CrossRef]
63. Reimann, J.; Brylski, M.; Aßmann, U. A tool-supported quality smell catalogue for android developers. In Proceedings of the Conference Modellierung 2014 in the Workshop Modellbasierte und Modellgetriebene Softwaremodernisierung–MMSM, Vienna, Austria, 19–21 March 2014. Available online: https://www.semanticscholar.org/paper/A-Tool-Supported-Quality-Smell-Catalogue-For-Reimann-Brylski/7ce532e0e933037c5f410a9e3cbc07f5513c3078 (accessed on 1 September 2021).
64. Johann, T.; Dick, M.; Naumann, S.; Kern, E. How to measure energy-efficiency of software: Metrics and measurement results. In Proceedings of the 2012 First International Workshop on Green and Sustainable Software (GREENS), Zurich, Switzerland, 3 June 2012; pp. 51–54.
65. Yang, X.; Han, M.; Tang, H.; Li, Q.; Luo, X. Detecting Defects With Support Vector Machine in Logistics Packaging Boxes for Edge Computing. *IEEE Access* **2020**, *8*, 64002–64010. [CrossRef]