The Evolution of Design Pattern Grime: An Industrial Case Study

Daniel Feitosa[1[0000-0001-9371-232X]], Paris Avgeriou[1[0000-0002-7101-0754]], Apostolos Ampatzoglou[1[0000-0002-5764-7302]], and Elisa Yumi Nakagawa[2]

[1]Department of Mathematics and Computer Science, University of Groningen, the Netherlands
[2]Department of Computer Systems, University of São Paulo, Brazil

d.feitosa@rug.nl, paris@cs.rug.nl, a.ampatzoglou@rug.nl, elisa@icmc.usp.br

Abstract. *Context:* GoF design patterns are popular among both researchers and practitioners, in the sense that software can be largely comprised of pattern instances. However, there are concerns regarding the efficacy with which software engineers maintain pattern instances, which tend to decay over the software lifetime if no special emphasis is placed on them. Pattern grime (i.e., degradation of the instance due to buildup of unrelated artifacts) has been pointed out as one recurrent reason for the decay of GoF pattern instances. *Goal:* Seeking to explore this issue, we investigate the existence of relations between the accumulation of grime in pattern instances and various related factors: (a) projects, (b) pattern types, (c) developers, and (d) the structural characteristics of the pattern participating classes. *Method:* For that, we empirically assessed these relations through an industrial exploratory case study involving five projects (approx. 260,000 lines of code). *Results:* Our findings suggest a linear accumulation of pattern grime, which may depend on pattern type and developer. Moreover, we present and discuss a series of correlations between the accumulation of pattern grime and structural characteristics. *Conclusions:* The outcome of our study can benefit both researchers and practitioners, as it points to interesting future work opportunities and also implications relevant to the refinement of best practices, the raise awareness among developers, and the monitoring of pattern grime accumulation.

Keywords: design patterns; pattern grime; industrial case study

1 Introduction

The most popular catalogue of design patterns among practitioners consists of the 23 GoF design patterns (from the Gang of Four—Gamma, Johnson, Helm, and Vlissides) [1]. In Java applications, it has been reported that the number of classes that participate in GoF pattern occurrences can vary from 15% to 65% (e.g., in software libraries) [2, 3], leading to a significant influence on the overall quality of the system. However, the effect of patterns on quality is not uniform [4]; the same pattern can

have both a positive and a negative effect on the quality of a software product. Therefore, gaining more insights on how exactly patterns have an impact on quality is of paramount importance. A significant parameter that determines how pattern instances affect quality is the amount of artifacts (e.g., methods and attributes) that exist in the pattern-participant classes, which however, are not compliant to the original pattern definition [5]. Izurieta and Bieman [5] named this phenomenon **pattern grime** and defined it as *the degradation of design pattern instance due to buildup of unrelated artifacts in pattern instances*. For example, grime can be introduced to a Template Method pattern instance by adding public methods that are not invoked inside the template method. Similarly, grime is introduced to a concrete state class of a State pattern instantiation when adding public methods other than those defined in the state interface. For both the aforementioned examples, such changes would lead to a reduced cohesion for the specific class, as well as reduced levels of source code understanding. Thus, the accumulation of grime can certainly be harmful to the quality of pattern instances and the overall system [5–7].

Despite the potential effect of pattern grime on software quality, there is currently a lack of studies that investigate factors related to the accumulation of pattern grime. Therefore, in this study, we take a first step by exploring two types of factors related to the accumulation of pattern grime, i.e., different: ***projects, pattern types, developers***, and ***structural characteristics of pattern-participating classes*** (e.g., coupling and lack of cohesion). To this end, we performed an industrial case study, in which we analyzed five projects (that sum up to approx. 260,000 source lines of code) containing eight different GoF pattern types and implemented by 16 developers. To measure grime, we provide an open-source tool that automates the assessment of several pattern grime metrics. The outcome of this study sheds light on the factors that influence the accumulation of grime in pattern instances. Our results can be used by architects and designers to develop best practices while using design patterns, but also to monitor the evolution of grime and its respective effect on software quality.

The remainder of this paper is organized as follows. In Section 2, we present work related to ours. The design of the case study is presented in Section 3, reported according to the guidelines of Runeson et al. [8], i.e., the Linear Analytic Structure. In Sections 4 and 5 we present the results of our study and discuss the most important findings, respectively. We report on the identified threats to validity and actions taken to mitigate them in Section 6. Finally, in Section 7 we conclude the paper and present some interesting extensions for this study.

2 Related Work

In this section, we present work reporting on empirical studies on the evolution of grime and / or its relation to other characteristics of software pattern instances (e.g., quality attributes and metrics).

Izurieta and Bieman [9] investigated the evolution of various design pattern instances from an open-source project to understand how patterns decay. The results suggest that the main reason for pattern instances to decay is due to grime. Schanz

and Izurieta [10] proposed a taxonomy for subtypes of modular grime (one type of grime) and performed a pilot study on nine pattern instances evolving throughout eight versions of one industrial software. The study validated the proposed classification, as well as suggested an increase of pattern grime. Regarding how the accumulation of grime correlates to other characteristics of the system, Griffith and Izurieta [11] proposed a taxonomy for one type of grime, class grime, and performed a pilot study on randomly selected pattern instances from open-source projects to investigate the effects of class grime on design pattern understandability, and found this quality attribute to be negatively affected by the accumulation of class grime. In another study, Izurieta and Bieman [6] evaluated the testability of design pattern instances from three different patterns and found that as grime is accumulated, other issues such as code smells also appears, and the testability of the pattern instances decreases.

Izurieta and Bieman [5] studied the accumulation of grime and rot (another form of pattern decay, due to deterioration of the structural or functional integrity) during the evolution of pattern instances of three open-source systems. The study also correlated grime to testability, adaptability and pattern instability. The results are similar to those observed in the aforementioned studies, including increase of pattern grime and negative correlation with testability and adaptability. The authors also reported that they could not identify rot of pattern instances nor correlation between grime and pattern instability. Dale and Izurieta [7] reported an experiment to study the correlation between three subtypes of modular grime and technical debt. Pattern instances of three example systems were used and modular grime was systematically injected in the instances. The results suggest that one subtype of modular grime (i.e., strength) is more strongly correlated to technical debt, in the sense that strong coupling (through class attributes) is correlated with stable grime, while weak coupling (other kinds of coupling) is correlated to increased technical debt.

In comparison to related work, we contribute the following: (a) we studied five industrial non-trivial projects that collectively provided 36,571 units of analysis (i.e., editions to pattern instances' source code, see Section 3). Therefore, we can compare our results with those obtained from the analysis of open-source projects and toy examples; (b) among other facets, we investigated how pattern grime is accumulated by different developers (16 in total), which has not been considered in previous studies; and (c) we studied the correlation between pattern grime and multiple structural metrics of pattern instances, which has not been thoroughly explored in previous studies.

3 Study Design

Objectives and Research Questions (RQs): The goal of this study, described using the Goal-Question-Metric (GQM) approach [12], is formulated as follows: "*analyze* instances of GoF design patterns *for the purpose of* investigating the factors of project, pattern type, developers and structural characteristics of pattern participants *with respect to* their relationship with pattern grime, *from the point of view of* software designers *in the context of* industrial software development". Based on this goal, we defined the following research questions—RQs:

RQ_1: How does grime accumulate in pattern instances?
$RQ_{1.1}$: Are there differences in accumulated grime among different projects?
$RQ_{1.2}$: Are there differences in accumulated grime among different pattern types?
$RQ_{1.3}$: Are there differences in accumulated grime among different developers?

RQ_1 aims at assessing pattern grime within the five projects and exploring differences across three different factors: projects, types of pattern (e.g., Observer, Template Method) and developers. We chose these factors as they may potentially influence the accumulation of grime: the projects vary in requirements, design, size, scope etc. and may thus influence grime accumulation; the types of patterns exhibit different solutions and may allow or inhibit the accumulation of grime; the developers have diverse backgrounds and experience thus knowingly or inadvertently accumulating grime differently.

RQ_2: Are structural characteristics of the pattern participants related to the accumulation of grime?

RQ_2 aims at investigating the relationship between levels of grime and a different type of factor: the structural characteristics of pattern-participating classes. This helps to further understand the details of how the structure of the pattern itself relates to accumulating grime, and can thus inform best practices on the usage of design patterns.

Case Selection, Unit of Analysis, and Subjects: To answer the research questions, we designed an exploratory case study [8], in which we analyze five industrial projects from one company in the domain of web and mobile applications development. Two projects were developed by two independent teams, whereas the remaining three projects were developed by a third team. We selected an industrial case study, since there is a lack of empirical studies on pattern grime for such projects; *most of the previous studies have been performed on toy examples or open-source projects*.

As cases, we used the pattern instances of the explored projects. From each case, we recorded multiple units of analysis, based on the evolution of the specific instance. In particular, we recorded a unit of analysis for every change in the instance (i.e., pair of successive commits). We decided to focus on pairs of commits to *isolate and assess events (changes to pattern instances) performed by a single developer*. This allows to investigate developers as a potential factor influencing grime (see RQ1.3).

Variables and Data Collection: To answer the research questions, we extracted four groups of variables:

1/ Identification of unit of analysis (commit, developer). To identify every unit of analysis, we queried the git repository and extracted the author information and files that were changed for every commit. We ignored merge commits, as they do not provide new information regarding changes to files. In addition, we considered only changes to java classes that participate in a pattern instance.

2/ Pattern information (instance-id, pattern). The collection of the pattern instances is a time-consuming task. For that reason, we used two tools, namely SSA (Similarity Score Analysis, v4.12) [13] and SSA+ (v1.0.0), to detect pattern instances and performed a series of validations. In short, these tools allow us to detect pattern in-

stances of 12 types: Adapter / Command, Composite, Decorator, Factory Method, Observer, Prototype, Singleton, State / Strategy, Template Method, and Visitor. Due to space limitations, we do not elaborate on the SSA tool nor its validation. However, we used a similar design setup to detect patterns in a previous study [14], in which all relevant information can be found. We note that we manually verified various (randomly selected) outputs. Regarding SSA+, it detects 10 extended pattern-participant classes, i.e., that participate in the pattern but it is not part of the main pattern structure (e.g., Concrete State / Strategy). The full list of detected extended pattern participants is available in the tool's website[1]. To validate SSA+, we also manually verified randomly selected outputs.

3/ *Assessment of grime change (cg-*, mg-*, og-*) between a pair of successive commits*. According to Izurieta and Bieman [9], there are three types of grime, which can be assessed independently: class, modular and organizational. To measure these types, we selected six metrics, as shown in Table I. Each metric is estimated based on diverse design elements of pattern-participating classes: (a) class grime metrics are based on attributes and public methods; (b) modular grime metrics are based on incoming and outgoing dependencies; and (c) organization grime metrics are based on package and their dependencies. Due to space limitations, we do not elaborate further on the metrics, which are calculated as described by Izurieta and Bieman [9]. We chose these metrics because they allow us to assess different aspects of each type of grime, and they were previously used and validated to assess pattern grime in non-trivial systems [5]. To automate the calculation of the metrics, we created an open-source tool, *spoon-pttgrime*[2] (v0.1.0), available online as a public repository, which also contains further information on how the metrics are calculated. To validate the tool, we manually verified the output for 20 pattern instances (randomly selected) over five consecutive commits. Bugs were fixed and additional verification rounds showed no errors. As we are interested in assessing the change of grime in pattern instances for a pair of commits, we subtracted the grime estimation at the current commit (identified by the unit of analysis) from the estimation of the immediate previous commit (i.e., its parent).

4/ *Assessment of structural change (s-*) between a pair of successive commits*. To assess structural change, we selected three sets of metrics, proposed by Chidamber and Kemerer [15], Li and Henry [16], and Bansiya and Davis [17], accounting for the 21 metrics presented in Table 1. We selected these metrics because they allow us to investigate many characteristics of the structure of pattern participants, and because they are well-known by both researchers and practitioners. To calculate the metrics, we used Percerons Client, i.e., a tool developed in our research group that automates the assessment of these metrics for Java classes. Percerons is a software engineering platform [18] to facilitate empirical research in software engineering and has been used for similar reasons in [11] and [19].

---

[1] https://github.com/search-rug/ssap

[2] https://github.com/search-rug/spoon-pttgrime

Presented in Table 1, these variables are recorded for each unit of analysis (i.e., change to pattern instance). The entire process of identifying and measuring the units of analysis culminates in the creation of a dataset of all extracted variables for each unit. This dataset is recorded as a table in which the columns correspond to collected variables. We clarify that due to a non-disclosure agreement signed with the company in this case study we cannot share the created dataset.

Analysis Procedure: To answer RQ_1, we analyze the descriptive statistics of the variables for unit identification, pattern information, and assessment of grime change. As our study comprises several projects / subjects and encompasses several GoF patterns, we derive data subsets, so as to group the units of analysis based on the different analyzed factors (i.e., project, pattern and developer). When necessary we also perform linear regressions and parametric or non-parametric tests [20] in order to devise trends and test differences between groups. To answer RQ_2, we first analyze whether the distribution of all measurements for each metric is normally distributed. If true, we can select the Pearson correlation method [20], otherwise the Spearman's rank correlation method [20]. For each pattern grime metric, we perform the analysis as follows: first we calculate the correlation between the grime metric and all structural metrics; next, we identify strong correlations (> 0.8) that are statistically significant, and discuss the results from the perspective of grime accumulation.

Table 1. List of collected variables

Group	Variable	Description
Unit Information	project	Project from which the pattern instance was extracted
	commit	Hash of the commit in the git repository
	dev	Developer author of the commit
Pattern Information	inst_id	ID of the pattern instance the class belongs to
	pattern	GoF design pattern of the instance
Assessment of Grime Change	cg-npm	Difference in the total number of alien public methods in all classes of the pattern instance (Class grime)
	cg-na	Difference in the total number of alien attributes in all classes of the pattern instance (Class grime)
	mg-ca	Difference in the pattern instance afferent coupling (Modular grime)
	mg-ce	Difference in the pattern instance efferent coupling (Modular grime)
	og-np	Difference in the number of packages within the pattern instance (Organizational grime)
	og-ca	Difference in the fan-in at the package level (Organizational grime)
Assessment of Structural Change	s-wmc	Difference in the average weighted methods per class
	s-dit	Difference in the maximum depth of inheritance tree
	s-noc	Difference in the average number of children
	s-cbo	Difference in the average coupling between object classes
	s-rfc	Difference in the average response for a class
	s-lcom	Difference in the average lack of cohesion in methods
	s-nom	Difference in the average number of methods
	s-mpc	Difference in the average message-passing coupling
	s-dac	Difference in the average data abstraction coupling

Group	Variable	Description
	s-size1	Difference in the lines of code
	s-size2	Difference in the number of properties
	s-dsc	Difference in the design size in classes
	s-noh	Difference in the number of hierarchies
	s-ana	Difference in the average number of ancestors
	s-dam	Difference in the data access metric
	s-camc	Difference in the cohesion among methods of class
	s-moa	Difference in the measure of aggregation
	s-mfa	Difference in the measure of functional abstraction
	s-nop	Difference in the number of polymorphic methods
	s-cis	Difference in the class interface size
	s-fan-in	Difference in the afferent couplings

4 Results

In this section, first we briefly describe the collected data and subsequently address each research question independently. We note that we investigated six metrics for pattern grime, and therefore report the results for all metrics and highlight findings independently for each one, when this is relevant. We collected a total of 1,422 commits, from the five studied projects, that include the creation or modification of pattern-participating classes. From these commits, 94% (i.e., 1,341) include modifications to one or more pattern instances. We identified 2,349 pattern instances of eight different GoF patterns: (Object) Adapter-Command, Factory Method, Observer, Singleton, State-Strategy, and Template Method. Each pattern instance was created and then modified up to 178 times (i.e., the maximum number of modifications for a single instance). From the total number of pattern instances, 87% (i.e., 2,039) were modified at least once after being created, and 64% (i.e., 1,500) at least five times. The data collection resulted in the identification of 36,571 units of analysis (i.e., creation / modification of a pattern instance in a commit).

RQ1 - Accumulation of Grime: To study the differences in accumulated grime among different projects, types of patterns and developers, we first present how the assessed pattern grime metrics change within the instances' evolution. Table 2 shows the following descriptive statistics for the six metrics (previously presented in Table 1): minimum and maximum values, mean value among all units of analysis and standard deviation (i.e., how much measurements vary from the mean value). Based on the Table 2, we notice that grime can either reduce (i.e., negative measurement) or increase. However, the data suggest that on average, grime in pattern instances tends to increase during the instance's evolution. Another observation is that the number of packages in a pattern instance (og-np) seems to be the grime indicator that is less likely to change, which is a probable sign of common design practices. Moreover, despite considerably higher maximum values, we notice that the measurements are consistently close to the mean, since the standard deviation is not much higher than the mean (especially compared to maximum values).

Table 2. Amount of grime accumulated per commit

Metric	Minimum	Maximum	Mean	Std. Deviation
cg-npm	-1.00	15.00	0.28	0.64
cg-na	-1.50	9.50	0.12	0.45
mg-ca	-3.75	44.00	0.21	1.18
mg-ce	-10.00	85.00	1.61	4.53
og-np	-0.25	2.00	0.02	0.14
og-ca	-2.00	35.00	0.14	1.13

Next, we are interested in investigating how grime accumulated in different projects ($RQ_{1.1}$). Fig. 1 depicts this information for the six metrics. P1 is the project with most collected commits (605), while P5 provided the least commits (76). The y-axis represents the mean amount of grime accumulated per modified instance in a given commit. The x-axis represents consecutive commits. We note that the x-axis does not represent the full history of commits. Our goal is to investigate the evolution of pattern instances and, thus, we considered only commits that include the modification of pattern-participant classes. By inspecting Fig. 1, we observe that every project individually reflects the trend of the population, i.e., pattern grime linearly increases during the project evolution. To verify this, we performed linear regression for every pair *<metric, project>* and assessed how well the calculated equation fits the data.

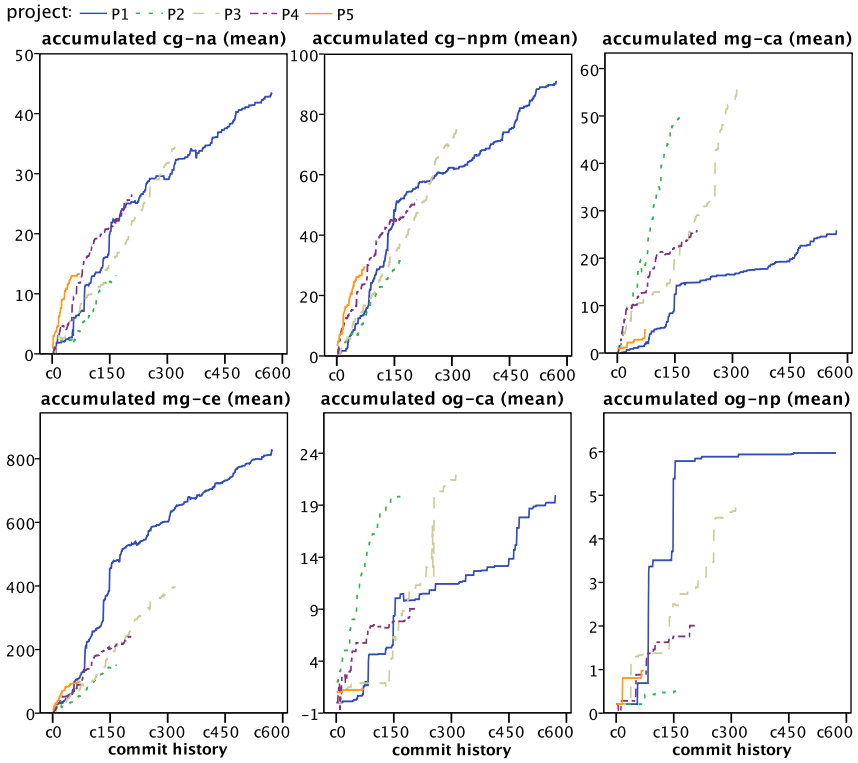

Fig. 1. Accumulation of grime per project for each grime metric

In Table 3, we present the results, which are all statistically significant. We notice that the vast majority of the equations are powerful descriptors, since R^2 (i.e., how close the data fit the regression line) is close to 1. The exceptions are the tuples *<og-np, P1>*, *<og-np, P5>*, and *<og-ca, P5>*, which regard metrics of organization grime. This is due to the drastic change in the accumulated grime observed for these tuples, which may reflect systematic changes in the design (e.g., package renaming).

Table 3. Linear regression of pattern grime accumulation per project

Metric	Project	Equation	R^2	Metric	Project	Equation	R^2
cg-npm	P1	13.91 + 0.15x	0.91	cg-na	P1	5.47 + 0.07x	0.93
	P2	-0.28 + 0.19x	0.99		P2	-0.59 + 0.08x	0.92
	P3	-1.79 + 0.24x	0.99		P3	-0.51 + 0.11x	0.99
	P4	7.44 + 0.24x	0.95		P4	1.89 + 0.13x	0.95
	P5	5.32 + 0.37x	0.95		P5	3.44 + 0.17x	0.89
mg-ca	P1	2.27 + 0.04x	0.90	mg-ce	P1	129.84 + 1.40x	0.89
	P2	-1.72 + 0.34x	0.99		P2	-9.04 + 1.00x	0.99
	P3	-2.24 + 0.17x	0.93		P3	-15.42 + 1.34x	0.99
	P4	5.68 + 0.11x	0.92		P4	15.36 + 1.21x	0.96
	P5	0.71 + 0.04x	0.87		P5	26.47 + 1.20x	0.89
og-np	P1	2.00 + 0.01x	0.58	og-ca	P1	1.09 + 0.03x	0.92
	P2	-0.06 + 0.00x	0.82		P2	3.11 + 0.12x	0.95
	P3	-0.20 + 0.02x	0.96		P3	-3.86 + 0.08x	0.90
	P4	-0.02 + 0.01x	0.89		P4	2.61 + 0.03x	0.81
	P5	0.12 + 0.01x	0.61		P5	1.04 + 0.00x	0.64

Further, we analyzed the dataset regarding different GoF patterns ($RQ_{1.2}$). In Table 4, we show the descriptive statistics for each metric and identified pattern. Due to space limitations, we do not report the results for the Observer and Template Method patterns, as the number of units of analysis for them is negligible (18 and 5, respectively). The results suggest that different patterns are subject to different levels of grime. For example, it seems that little grime is accumulated in instances of Singleton after their creation, whilst instances of Factory Method tend to accumulate the most amount of grime. To statistically investigate the difference between patterns, we performed pairwise comparisons (Mann-Whitney tests), which, due to lack space, are reported within the supplementary material [21]. The results showed that the differences in most comparisons (86% of the 36 tests) is statistically significant, thus supporting our findings.

Table 4. Amount of grime accumulated per pattern

Metric	Pattern	Num. of instances	Num. of changes	Min.	Max.	Mean	Std. Deviation
cg-na	Adapter-Command	770	13,225	-3.00	17.00	0.12	0.53
	Factory Method	61	776	-3.42	13.00	0.15	0.78
	Singleton	83	281	-1.00	1.00	0.01	0.16
	State-Strategy	1121	19,937	-4.00	13.00	0.10	0.44

Metric	Pattern	Num. of instances	Num. of changes	Min.	Max.	Mean	Std. Deviation
cg-npm	Adapter-Command	770	13,225	-3.00	26.00	0.21	0.77
	Factory Method	61	776	-7.58	21.67	0.35	1.42
	Singleton	83	281	-2.00	4.00	0.06	0.44
	State-Strategy	1121	19,937	-8.00	21.33	0.21	0.80
mg-ca	Adapter-Command	770	13,225	-2.00	44.00	0.08	0.89
	Factory Method	61	776	-7.00	102.00	0.59	4.15
	Singleton	83	281	-1.00	7.00	0.49	0.96
	State-Strategy	1121	19,937	-15.00	44.00	0.12	0.87
mg-ce	Adapter-Command	770	13,225	-20.00	197.00	1.19	5.60
	Factory Method	61	776	-13.00	60.00	1.44	4.40
	Singleton	83	281	-4.00	17.00	0.47	1.85
	State-Strategy	1121	19,937	-30.00	159.00	1.23	5.66
og-ca	Adapter-Command	770	13,225	-2.00	35.00	0.06	0.75
	Factory Method	61	776	-36.00	36.00	0.17	2.32
	Singleton	83	281	-1.00	27.00	0.41	2.19
	State-Strategy	1121	19,937	-6.00	34.00	0.06	0.62
og-np	Adapter-Command	770	13,225	0.00	2.00	0.01	0.13
	Factory Method	61	776	-1.00	3.00	0.03	0.21
	Singleton	83	281	0.00	1.00	0.01	0.10
	State-Strategy	1121	19,937	-1.00	3.00	0.01	0.15

The last facet we investigated was how different developers accumulate grime ($RQ_{1.3}$). Due to space limitations, we do not report the complete descriptive statistics for each metric and developer, which are available within the supplementary material [21]. In Table 5, we present the number of pattern instances maintained by the 16 developers, changes to pattern instances and mean value of the grime metrics. By analyzing the results, we notice that some developers seem to consistently accumulate more grime than others (e.g., D7, D8 and D9), or less grime than others (e.g., D1 and D3), with respect to most metrics. Furthermore, we can observe that developers that changed pattern instances more often tend to accumulate less grime. Seeking to support our observations statistically, we compared pairs of developers based on every metric using the Mann-Whitney test. By observing the findings of the test, we suggest that 73% of the 396 tests are statistically significant, and that the non-significant tests regard mostly the number of packages (og-np). Due to lack space, detailed results are reported on the supplementary material [21].

Table 5. Average amount of grime accumulated per developer

Developer	Num. of instances	Num. of changes	cg-na	cg-npm	mg-ca	mg-ce	og-ca	og-np
D1	465	7,525	0.08	0.17	0.04	0.93	0.04	0.00
D2	1,132	6,232	0.12	0.34	0.20	1.25	0.05	0.00
D3	549	5,232	0.07	0.07	0.04	0.85	0.01	0.00
D4	837	5,141	0.10	0.14	0.13	0.96	0.04	0.01
D5	335	3,442	0.10	0.23	0.04	1.35	0.02	0.02

Developer	Num. of instances	Num. of changes	cg-na	cg-npm	mg-ca	mg-ce	og-ca	og-np
D6	469	1,554	0.13	0.24	0.14	2.28	0.24	0.00
D7	292	1,406	0.17	0.29	0.23	2.54	0.19	0.05
D8	326	1,346	0.20	0.26	0.21	1.72	0.18	0.02
D9	161	697	0.13	0.38	0.27	1.68	0.20	0.02
D10	225	636	0.07	0.37	0.24	1.05	0.27	0.01
D11	233	515	0.01	0.19	0.34	0.37	0.06	0.00
D12	170	431	0.23	0.28	0.06	1.89	0.00	0.00
D13	41	56	0.79	1.64	0.89	8.04	0.29	0.21
D14	13	17	0.00	0.00	0.00	2.06	0.00	0.00
D15	3	8	0.00	0.03	-0.25	0.00	0.38	0.00
D16	2	4	0.00	0.00	0.00	0.75	0.00	0.00

Summarizing the results for RQ_1, pattern grime: (a) *is likely to increase linearly over system evolution*; (b) *grows similarly across different projects*; (c) *accumulates at different paces depending on the pattern type and the individual developer*. The interpretation of all findings reported in this section, as well as their implications to researchers and practitioners are discussed in Section 5.

RQ2 - Structural Characteristics and Pattern Grime: To assess the correlation between pattern grime and structural metrics, we first verified whether all measurements for each metric are normally distributed. We found that not all are normally distributed and, thus, we decided to use a non-parametric method to study the metrics: Spearman's rank correlation. All assessed correlations are presented in Table 6, and are all statistically significant.

Table 6. Correlation between grime and structural metrics

	cg-npm	cg-na	mg-ca	mg-ce	og-np	og-ca
s-wmc	*0.86*	0.44	0.38	0.48	0.46	0.38
s-dit	0.44	0.53	0.55	0.43	*0.99*	0.71
s-noc	0.45	0.52	0.60	0.41	*0.99*	0.73
s-cbo	0.47	0.67	0.50	0.73	0.46	0.41
s-rfc	0.65	0.54	0.31	0.65	0.41	0.33
s-lcom	0.70	0.35	0.32	0.36	0.35	0.31
s-nom	*0.86*	0.44	0.38	0.48	0.46	0.38
s-mpc	0.43	0.44	0.22	0.55	0.35	0.27
s-dac	0.36	*0.87*	0.34	0.59	0.56	0.42
s-size1	0.69	0.53	0.31	0.58	0.41	0.35
s-size2	0.79	0.65	0.38	0.58	0.44	0.38
s-dsc	0.44	0.53	0.56	0.43	*0.99*	0.70
s-noh	0.38	0.43	0.50	0.35	0.77	0.60
s-ana	0.45	0.53	0.51	0.43	*0.93*	0.66
s-dam	0.34	0.56	0.42	0.43	0.76	0.54
s-camc	-0.14	0.16	0.17	0.03	0.45	0.30
s-moa	0.37	*0.90*	0.36	0.61	0.58	0.44

	cg-npm	cg-na	mg-ca	mg-ce	og-np	og-ca
s-mfa	0.03	0.11	0.03	0.08	0.21	0.07
s-nop	0.71	0.35	0.48	0.34	0.51	0.43
s-cis	*0.97*	0.41	0.41	0.44	0.48	0.41
s-fan-in	0.46	0.42	*0.90*	0.36	0.70	0.63

Regarding the metrics for ***class grime***, we make the following observations. The metric cg-npm is strongly correlated (> 0.8) to s-wmc, s-nom, and s-cis. This may be an indication that when many methods are added to pattern-related classes it is common that a large portion of them are not related to the pattern realization. The metric cg-na is strongly correlated to s-dac and s-moa. This may be an indication that a considerable part of the pattern instance coupling is coming from added attributes. This may not be necessarily an alert for bad design, but it rather depends on how many attributes are added. Regarding ***modular grime***, we notice that the metric mg-ca is strongly correlated to s-fan-in only, which is a metric that is similar to mg-ca, but at class level. This suggests that most of the pattern instance afferent coupling comes from regular afferent coupling of the pattern participants. This may indicate that pattern instances tend to evolve by adding functionality not related to the pattern. The metric mg-ce is not strongly correlated to any metrics, whereas the strongest correlations are with s-rfc and s-cbo. These moderate correlations also indicate that, to some extent, the introduction of coupling in pattern instances is also introducing grime. Finally, regarding ***organizational grime***, the metric og-np is strongly correlated to s-dit, s-noc, s-dsc, and s-ana. Despite the strong correlations, this finding may be inconclusive as og-np rarely changes and this is probably the main reason for such high correlations. Finally, the metric og-ca is not strongly correlated to any metrics, whereas the strongest correlations are with s-noc, s-dit and s-dsc. These moderate correlations may indicate that, to some extent, the addition of new classes to the pattern instance is to serve a new purpose, i.e., serve a class not served before.

5 Discussion

In this section, we discuss the findings of our case study, as well as their implications. First, we interpret our findings, elaborating on explanations and consequences for the observed results. Next, we present how our findings can benefit both researchers and practitioners.

Interpretation of Results: In Section 4, we reported the raw findings of our case study, whereas in this section, we interpret them and compare them against the state-of-the-art. First, regarding the evolution of grime, we observed that ***pattern grime is constantly increasing along the versions of a system***. This result can be considered intuitive as it aligns with Lehman's laws on software evolution: software quality deteriorates as the software becomes larger and more complex. However, there is an interesting aspect of this finding: the amount of grime that is accumulated in pattern instances clearly suggests that pattern-participating classes are not "closed to modifications", in the sense that they are continuously "polluted" with artifacts (e.g., methods,

dependencies, etc.) that are not pattern-related. This pollution potentially influences how the application of design patterns affects quality attribute indicators of a system. Thus, pattern instantiation does not have a constant effect on quality, but it changes along evolution. This finding is in accordance to the literature, which suggests that the effect of GoF design patterns on product quality is not uniform along different pattern instances [4], and aligns with results of studies with similar setups [5–7]. In particular, Izurieta and Bieman [5] used the same pattern grime metrics and investigated some patterns in common (e.g., Singleton and Factory Method), but by inspecting open-source systems. The results of both studies agree on the increase of grime metrics.

Regarding the three parameters that were investigated in RQ_1 (i.e., grime in different projects, patterns, and developers), the results suggested that ***the levels of grime are similar at the different projects of the same company despite the little overlap of developers among projects***. This outcome can be potentially explained by the fact that the developers were guided by the same practices, since they usually follow the same company process. Nevertheless, this finding needs to be further validated through a follow-up study conducted in different companies. Another finding is that ***the levels of grime are different among pattern types***, which complies with the literature suggesting that different patterns have different effect on quality attributes (e.g., [3] on stability). In particular, we noticed that instances of the Singleton pattern are the least likely to accumulate grime, whereas instances of Factory Method are the most grime-prone. The acknowledgement of certain good practices (e.g., avoid creation of God Classes) can lead to more "grime-free" Singleton instances. However, if not careful, developers may enlarge the responsibility of classes unnecessarily, as observed with Factory Method instances, which may include methods that suffer from the Feature Envy, Shotgun Surgery, or Divergent Change smells. Therefore, *we suggest monitoring pattern grime to identify spots of bad quality in the system*. Such a practice may support the preservation of quality indicators (such as understandability and testability) at acceptable levels and thus increase productivity. Moreover, in comparison with related work, Izurieta and Bieman [5] also show that Singleton pattern instances tend to accumulate less grime, whereas on the contrary Factory Method instances tend to accumulate grime faster than other investigated patterns. This observation further supports that open-source and industrial systems have similarities with regards to the accumulation of pattern grime.

From the last investigated parameter, we found that ***the levels of grime also differ among developers***. Their tendency to accumulate grime likely depends on diverse factors. In particular, varied levels of programming skills, knowledge of the system and of GoF patterns can explain the different tendency to accumulate grime. This finding supports the belief that personalized quality assessments are required in industry [22]. Furthermore, we observed that developers that performed more changes are related to lower levels of accumulated grime, suggesting that most tasks (resulting in more changes) are assigned to more experienced developers, inclined to accumulate less grime. *We suggest using such information about developers in order to improve the software development process*. For example, since our industrial partner use agile methodologies, such information can be considered in daily Scrum meetings in which issues are assigned to individual developers. The personalization of software devel-

opment and the effect on human factors in the quality of the software have been extensively studied in the last years, underlying the importance of such strategies.

Finally, regarding the relation of structural metrics with grime metrics, the results point out that some of the most established structural quality metrics are related to the grime metrics. For example, the fan-in metric is at least moderately correlated to all grime metrics. This finding may be explained by the fact that pattern grime is calculated at the detailed-design level. Since class dependencies consist one of the main elements of object-oriented design, it is intuitive to expect the obtained correlations, e.g., between two metrics that are calculated based on class dependencies. However, we note that the strength of the correlations varies among pattern instances, which shows that structural metrics can be adequate predictors of grime accumulation.

Implications to Researchers and Practitioners: *Researchers* can benefit from our results from several perspectives. We presented a thorough exploration of the accumulation of pattern grime and we identified several factors that influence how pattern grime grows during the evolution of pattern instances. This exploration not only reinforces the importance of investigating pattern grime, but also suggests several opportunities of future work, e.g., investigate characteristics of developers that tend to accumulate grime. In addition, the identified correlations between pattern grime and structural metrics help on understanding how pattern grime is introduced, as well as open further possibilities to investigate other relevant aspects of software systems and processes, for example technical debt. We also foresee benefits of our results to *practitioners*. Because we investigated five non-trivial projects, our findings can help practitioners improve best practices on the usage of design patterns, e.g., by warning developers to avoid accumulating grime on Singletons pattern instances. Moreover, the metrics and correlations that we present can be considered in processes for monitoring the evolution of the software systems, e.g., high levels of fan-in in pattern-participating classes may indicate that considerable grime is being inserted.

6 Threats to Validity

In this section, we discuss threats to construct validity (i.e., if the studied phenomenon is connected to the set objectives), reliability (if the study can be replicated), and external validity (i.e., generalizability). We do not analyze internal validity, as we do not try to establish causal relationships.

Concerning construct validity, the tool SSA is limited by its precision and recall: false positives and negatives may bias the presented results. However, to the best of our knowledge the used tool is among the most reputed in the community, and has adequate performance (see Section 3). For mitigating this threat, we verified its precision and recall manually by checking 30 random pattern instances for each GoF pattern that was detected (i.e., over 100 instances in total), which were all successful. Additionally, regarding SSA+ and spoon-pttgrime, we acknowledge that the tools may have bugs. However, we verified over 50 random outputs of each tool and, to the best of our knowledge, no bugs were found.

In order to mitigate reliability threats, two different researchers performed the collection and analysis, double-checking sample outputs. Besides that, we acknowledge that non-disclosure agreements do not allow us to share the collected dataset. However, all used tools are freely available and replication studies can be carried out. Finally, the external validity of our study is threatened by the fact that we analyzed projects of the same company, thus, our findings may not be generalizable to other projects nor teams. However, our results relate to those obtained in other studies with similar setup, e.g., we expected modular grime to be the main contributor for the pattern grime, and we found mg-ce to clearly grow at a faster pace. In addition, our results are bounded by our study design. Adding other GoF patterns, pattern grime metrics, or structural metrics could lead to adjustments in our findings.

7 Conclusion

In this paper, we presented an exploratory case study on how grime accumulates in pattern instances and its correlation to structural characteristics of the pattern participants. To this end, we investigated the evolution of 2,349 pattern instances of eight patterns, assessing six grime metrics of three types of grime (class, modular and organization), as well as 21 structural metrics. We explored how grime is distributed according to: (a) projects, pattern types, and developers, and (b) structural characteristics of pattern-participating classes. The results suggest that pattern grime tends to increase linearly, it is likely independent of project but depends on pattern type and developer. Moreover, we identified a series of correlations between metrics for pattern grime and structural characteristics, e.g., the coupling added to pattern participants tend to also introduce grime. Based on our results and observations, we envisage several opportunities for future work. First, some of the investigated facets on how pattern grime accumulates can and should be further explored, e.g., what factors may be related to developers that tend to accumulate more or less grime. Furthermore, our observations based on the correlation between pattern grime and structural metrics raised questions that can be investigated in confirmatory studies, e.g., whether most introduced afferent coupling is indeed resulting in the accumulation of grime.

Acknowledgements. The authors would like to thank the financial support from the Brazilian and Dutch agencies CAPES/Nuffic (Grant N.: 034/12), CNPq (Grant N.: 204607/2013-2), as well as INCT-SEC (Grant N.: 573963/2008-8 and 2008/57870-9).

References

1. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc. (1995).
2. Khomh, F., Gueheneuc, Y.-G., Antoniol, G.: Playing roles in design patterns: An empirical descriptive and analytic study. In: 25th IEEE International Conference on Software Maintenance. pp. 83–92. IEEE (2009).
3. Ampatzoglou, A., Chatzigeorgiou, A., Charalampidou, S., Avgeriou, P.: The Effect of GoF Design Patterns on Stability: A Case Study. IEEE Trans. Softw. Eng. 41, 781–802 (2015).

4. Ampatzoglou, A., Charalampidou, S., Stamelos, I.: Research state of the art on GoF design patterns: A mapping study. J. Syst. Softw. 86, 1945–1964 (2013).
5. Izurieta, C., Bieman, J.M.: A multiple case study of design pattern decay, grime, and rot in evolving software systems. Softw. Qual. J. 21, 289–323 (2013).
6. Izurieta, C., Bieman, J.M.: Testing Consequences of Grime Buildup in Object Oriented Design Patterns. In: First International Conference on Software Testing, Verification, and Validation. pp. 171–179. IEEE (2008).
7. Dale, M.R., Izurieta, C.: Impacts of design pattern decay on system quality. In: Eighth ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–4. ACM Press, New York, New York, USA (2014).
8. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Wiley Blackwell (2012).
9. Izurieta, C., Bieman, J.M.: How Software Designs Decay: A Pilot Study of Pattern Evolution. In: First International Symposium on Empirical Software Engineering and Measurement. pp. 449–451. IEEE (2007).
10. Schanz, T., Izurieta, C.: Object oriented design pattern decay. In: Fourth ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1--8. ACM Press, New York, New York, USA (2010).
11. Griffith, I., Izurieta, C.: Design pattern decay: the case for class grime. In: Eighth ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–4. ACM Press, New York, New York, USA (2014).
12. Basili, V.R., Caldiera, G., Rombach, H.D.: Goal Question Metric paradigm. In: Encyclopedia of Software Engineering. pp. 528–532. Wiley & Sons (1994).
13. Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., Halkidis, S.T.: Design pattern detection using similarity scoring. Softw. Eng. IEEE Trans. 32, 896–909 (2006).
14. Feitosa, D., Alders, R., Ampatzoglou, A., Avgeriou, P., Nakagawa, E.Y.: Investigating the effect of design patterns on energy consumption. J. Softw. Evol. Process. 29, e1851 (2017).
15. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20, 476–493 (1994).
16. Li, W., Henry, S.: Object-oriented metrics that predict maintainability. J. Syst. Softw. 23, 111–122 (1993).
17. Bansiya, J., Davis, C.G.: A hierarchical model for object-oriented design quality assessment. IEEE Trans. Softw. Eng. 28, 4–17 (2002).
18. Ampatzoglou, A., Michou, O., Stamelos, I.: Building and mining a repository of design pattern instances: Practical and research benefits. Entertain. Comput. 4, 131–142 (2013).
19. Alhusain, S., Coupland, S., John, R., Kavanagh, M.: Towards machine learning based design pattern recognition. In: 13th UK Workshop on Computational Intelligence. pp. 244–251. IEEE (2013).
20. Field, A.: Discovering Statistics Using SPSS. SAGE Publications Ltd (2009).
21. Feitosa, D., Avgeriou, P., Ampatzoglou, A., Nakagawa, E.Y.: Supplementary Material: "The Evolution of Design Pattern Grime: An Industrial Case Study," https://doi.org/10.5281/zenodo.806800.
22. Amanatidis, T., Chatzigeorgiou, A., Ampatzoglou, A., Stamelos, I.: Who is producing more technical debt? A personalized assessment of TD principal. In: Nineth International Workshop on Managing Technical Debt. pp. 1--8. ACM (2017).