Information and Software Technology 66 (2015) 58–72

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

freepaper.me FREE paper

Quantitative analysis of fault density in design patterns: An empirical study

Mahmoud O. Elish *, Mawal A. Mohammed

Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

ARTICLE INFO

Article history:
Received 31 October 2014
Received in revised form 24 May 2015
Accepted 26 May 2015
Available online 1 June 2015

Keywords:
Design patterns
Software quality
Fault density
Quantitative analysis

ABSTRACT

Context: There are many claimed advantages for the use of design patterns and their impact on software quality. However, there is no enough empirical evidence that supports these claimed benefits and some studies have found contrary results.
Objective: This empirical study aims to quantitatively measure and compare the fault density of motifs of design patterns in object-oriented systems at different levels: design level, category level, motif level, and role level.
Method: An empirical study was conducted that involved five open-source software systems. Data were analyzed using appropriate statistical test of significance differences.
Results: There is no consistent difference in fault density between classes that participate in design motifs and non-participant classes. However, classes that participate in structural design motifs tend to be less fault-dense. For creational design motifs, it was found that there is no clear tendency for the difference in fault density. For behavioral design motifs, it was found that there is no significant difference between participant classes and non-participant classes. We observed associations between five design motifs (Builder, Factory Method, Adapter, Composite and Decorator) and fault density. At the role level, we found that only one pair of roles (Adapter vs. Client) shows a significant difference in fault density.
Conclusion: There is no clear tendency for the difference in fault density between participant and non-participant classes in design motifs. However, structural design motifs have a negative association with fault density. The Builder design motif has a positive association with fault density whilst the Factory Method, Adapter, Composite, and Decorator design motifs have negative associations with fault density. Classes that participate in the Adapter role are less dense in faults than classes that participate in the Client role.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Design Patterns (DPs) are generic solutions to common design problems. The objective of cataloging these solutions, including their intent, motivation, applicability, structure, participants, consequences, etc. is to make them reusable. Gamma et al. [14] classified DPs (known as GoF DPs) into three categories: creational patterns, structural patterns, and behavioral patterns. Creational patterns are concerned with creating collections of objects in flexible ways. Structural patterns are concerned with representing collections of related objects. Behavioral patterns are concerned with capturing behavior among collections of objects. There are 23 GoF DPs: five creational patterns, seven structural patterns, and 11 behavioral patterns. Design motifs refer to the solution parts of DPs that are disseminated in the source code of the systems in which DPs are applied [28,39]. In a design motif, there is one or more participant classes that play different roles.

Since their introduction, DPs have attracted the attention of software researchers and practitioners due to the claimed advantages of their application. They are claimed to improve programmers' productivity, promote best design practices, help novice designers to acquire more experience in software design, and make communication easier among team members. Despite these claims, the impact of DPs on software quality is still a debatable issue. Zhang and Budgen [49] conducted a survey of experienced users' perceptions about DPs to determine which patterns do expert pattern users consider useful or not useful for software development and maintenance. They found that only three

* Corresponding author. Tel.: +966 13 8601150.
E-mail addresses: elish@kfupm.edu.sa (M.O. Elish), mawal.mohammed@yahoo.com (M.A. Mohammed).

http://dx.doi.org/10.1016/j.infsof.2015.05.006
0950-5849/© 2015 Elsevier B.V. All rights reserved.

patterns (Observer, Composite, and Abstract Factory) were widely regarded as valuable. Zhang and Budgen [50] also conducted a mapping study to determine the scale and extent to which empirical studies have been undertaken to evaluate the effectiveness of DPs. They concluded that DPs have been subjected to limited empirical evaluation and that more empirical evidence is very much needed. The need for further investigations on the impact of DPs on different software quality attributes are justifiable by the following reasons: (i) there is no consensus among the conducted studies in the literature, as discussed in the following section, on the impact of DPs on the different quality attributes; (ii) limited number of quality attributes have been addressed; (iii) not all DPs have been addressed; and (iv) not all levels (i.e., design level, category level, motif level and role level) have been evaluated.

One of the common arguments for the applications of DPs often relate to reducing the number of software faults [23,47]. The fault density of an object-oriented class is a measure of the number of confirmed and detected faults in the class divided by its size. Class size is usually positively correlated to the number of faults, which known as the confounding effect of class size [14]. Size measures are often used to normalize fault counts when evaluating quality, as in measures of fault density [30]. The main objective of this study is to quantitatively measure and compare the fault density of design motifs in object-oriented systems. For this purpose, we conducted an empirical study that:

- Measures and compares the fault density of participant versus non-participant classes in the design motifs.
- Measures and compares the fault density of participant classes across the different categories of design motifs in the GoF's book (creational, structural, and behavioral).
- Measures and compares the fault density of participant classes across design motifs.
- Measures and compares the fault density of participant classes across the different roles in each design motif.

Quantitative analysis of fault density in design motifs provides software developers with valuable knowledge of which motifs require special attention in their implementation and testing. This knowledge serves as recommendations and guidelines for software designers to effectively apply design motifs which, in turns, help in producing better designs. Moreover, software testers can utilize this knowledge to focus on the potentially troublesome parts of designs that require more attention. Therefore, software testers can write more useful test cases that address real design issues.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 discusses the empirical study setup. Section 4 presents the results of the empirical study, which are then analyzed and discussed in Section 5. Section 6 discusses threats to validity. Finally, Section 7 provides concluding remarks and directions for future work.

2. Related work

There are many studies in the literature that have explored the relationship between DPs and software quality attributes. A comprehensive literature survey was conducted by Ali and Elish [2] on the impact of DPs on quality attributes. Only four quality attributes have been investigated in the literature: maintainability, change-proneness, performance and energy consumption, and fault-proneness.

Prechelt et al. [42] conducted a controlled experiment to study the relationship between DPs and maintainability. They found that the use of DPs improves software maintainability. Vokac et al. [48] conducted a replication of this experiment. They found that different patterns have different impact on maintainability. Another four replications for this experiment were conducted in several universities [42]. In each replication, different findings on the impact of DPs on maintainability were reported: negative impacts [27,31,41] and no impact [36]. Garzas et al. [16] investigated the impact of three different DPs on maintainability as well. In their experiment, maintainability was measured in terms of understandability and modifiability. They found that DPs make design diagrams more difficult to understand and consequently require more time to modify. Finally, Hegedus et al. [25] performed a case study on a software system to evaluate the impact of DPs on maintainability. The reported findings show an improvement in maintainability with the use of DPs.

Aversano et al. [6] conducted a case study to address the impact of DPs on change-proneness. They found that the impact of DPs on change-proneness depends on the role of DPs in the functionality of the system (i.e., if DPs are involved in the implementation of major functionalities of the system, they will be subject to more changes). Bieman et al. [8] conducted another study to evaluate the impact of DPs on change-proneness at the class-level. They found that participant classes in design motifs are more change-prone than non-participant classes. Gatrell et al. [18] conducted a case study to evaluate the change-proneness of participant versus non-participant classes as well. They found that some DPs are associated with more changes than others. Posnett et al. [40] studied the influence of DPs roles on change-proneness. They found that the observed associations between change-proneness and the roles in patterns might be due to the sizes of the classes playing those roles. Penta et al. [39] reported an empirical study that investigated the relationship between DPs roles and the frequency/kind of changes. The results confirmed the intuitive behavior about changeability of many roles. Khomh et al. [28] conducted an empirical descriptive and analytic study of classes playing zero, one, or two roles in six different DPs. They found a significant increase in many internal metric values for classes playing two roles. They also found a significant increase in the frequencies and the number of changes of classes playing two roles.

Rudzki [43] conducted a study to evaluate the impact of two DPs, Command and Façade, on performance. He found that Façade performed better than Command. Afacan [1] performed an experiment to study the impact of State design pattern on memory usage and execution time. He found that a design with the State pattern consumes more resources than a design without DPs. However, the DP solution leads to clearer system architecture that can help improve other quality attributes. Some studies have investigated the energy consumption of DPs [9,33,44]. Bunse and Stiemer [9] presented a case study that examined the impact of DPs application onto a systems energy consumption. They found that the Decorator pattern has a negative impact on the energy needs of an app. Sahin et al. [44] presented a preliminary empirical study that investigates the impacts on energy usage of applying DPs. They found that applying DPs can both increase and decrease the amount of energy used by an application, and also DPs within a category do not impact energy usage in similar ways. Litke et al. [33] observed that the use of DPs does not necessarily impose a significant penalty on power consumption.

The relationship between DPs and faults has been investigated in three works. Vokac [47] investigated fault frequency (normalized number of faults over time) in DPs. Five patterns were investigated in his study: Singleton, Template Method, Decorator, Observer and Factory Method. He found that the DPs that are associated with larger structures, such as Observer and Singleton, are subject to more faults whereas Factory Method is loosely coupled

freepaper.me FREE paper

to other components of the system and is less subject to faults. For Template Method and Decorator, he observed no clear tendency. Gatrell and Counsell [17] studied the relationship between fault-proneness and DPs. Ten GoF's patterns were addressed in their study. They observed that pattern-based classes were more fault-prone than non-pattern classes. In particular, they found that Adapter, Method, and Singleton are more fault-prone than the other pattern classes, in terms of the numbers/sizes of changes made to a class to fix a fault. Ampatzoglou et al. [3] conducted a case study on software games to evaluate the impact of 11 DPs on defect frequency. They observed that Abstract Factory, Composite, Observer, State, Strategy, Prototype, and Proxy are negatively correlated to fault frequency. However, they observed that Adapter and Template Method are positively correlated to fault frequency. The study conducted by Ampatzoglou et al. [3] is the only study that was conducted on Java systems. The other two studies (i.e., [17,47] that addressed faults) were conducted on C# and C++ systems, respectively.

This research study is different from the previous work in many aspects. In this study, we address fault density, which has not been addressed in the literature. In addition, we target 17 patterns whilst the maximum number of the examined patterns in the literature is 11 [3,17,47]. Furthermore, we evaluate fault density at all levels: design level, category level, motif level, and role level. Table 1 shows a summary of the difference between this paper and previous works that have addressed the relationship between DPs and faults.

3. Empirical study setup

This section presents the empirical study setup. We first state the research questions and hypotheses. Then, we describe data collection. Finally, we discuss the statistical tests.

3.1. Research questions

To accomplish the objectives of this work, we quantitatively analyzed the fault density of design motifs at four granularity levels: design, category, motif, and role. Based on these granularity levels, we identified the following research questions.

- **RQ1:** Is the fault density of classes participating in design motifs significantly different to that of classes not participating in design motifs?
- **RQ2:** Are there significant differences in fault density among classes that participate in the different categories of design motifs?
- **RQ3:** Is the fault density of classes participating in a design motif significantly different to that of classes not participating in that design motif?
- **RQ4:** Are there significant differences in fault density among classes that participate in the different roles in each design motif?

3.2. Research hypotheses

To answer the research questions, we tested the following null hypotheses. The alternative hypotheses are the opposite of their corresponding null hypothesis, i.e., they state that "there is significant difference in fault density..."

Hypothesis 1. There is *no* significant difference in fault density between classes that participate in design motifs and those that do not.

Hypothesis 2. There is *no* significant difference in fault density among classes that participate in the different categories of design motifs.

Hypothesis 3. There is *no* significant difference in fault density between classes that participate in a design motif and those that do not participate in that motif.

Hypothesis 4. There is *no* significant difference in fault density among classes that participate in the different roles in each design motif.

3.3. Data collection

We conducted our empirical investigation on the following five open-source software systems:

- *JHotDraw* v5.1 is a framework for the creation of drawing editors. The creation of geometric and user defined shapes, editing those shapes, creating behavioral constraints in the editor and animation are supported by this framework.
- *JUnit* v3.7 is a simple framework for creating test cases that are used repeatedly. It is used to write test cases for Java programs.
- *Lexi* v0.1.1 alpha is a word processor. It can be used in editing many files, such as RTF and HTML files in addition to the plain text.
- *Nutch* v0.4 is an extensible and scalable Web crawler. It can be used in searching, indexing, and scoring of filers.
- *PMD* v1.8 is a source code analyzer. It scans the source code searching for standard coding rules violations and other problems, such as suboptimal code and dead code.

The reason for choosing these systems is the availability of their design motifs data in the P-Mart repository [19]. We could have used bigger systems other than those in the P-Mart repository, such as Eclipse, but the problem is with the collection of the design motifs data of these systems. Before deciding to use the P-Mart repository, we surveyed the literature searching for software design motifs detection tools. As a result, we identified 10 tools [5,13,20,24,26,34,37,38,45,46]. Then, we evaluated the suitability of these tools to our work. We found that these tools are not suitable for several reasons. First, all of these tools detect only a small

Table 1
Comparison with previous work.

Reference	Fault aspect	# Of examined DPs	Level(s) of analysis
Vokac [47]	Fault frequency	5	Motif
Gatrell and Counsell [17]	Numbers/sizes of changes to fix faults	10 + 3 non-GoF	Design and motif
Ampatzoglou et al. [3]	Fault frequency	11	Motif
This paper	Fault density	17	Design, category, motif, and role

freepaper.me FREE paper

Table 2
Comparison of design motifs detection tools.

Tools	# Of detectable design motifs	Precision (%)	Recall (%)
DeMIMA [24]	13	34	100
DP-Miner [26]	6	91–100	97
DPRE [34]	6	62–97	–
FUJABA [37]	23	–	–
MARRPLE [5]	3	78.6	78.3
Pinot [38]	17	–	–
Ptidej [20]	19	–	–
Tsantalis et al.'s tool [46]	12	100	95.9
SPQR [45]	1	–	–
WOP [13]	4	57.3	54.5

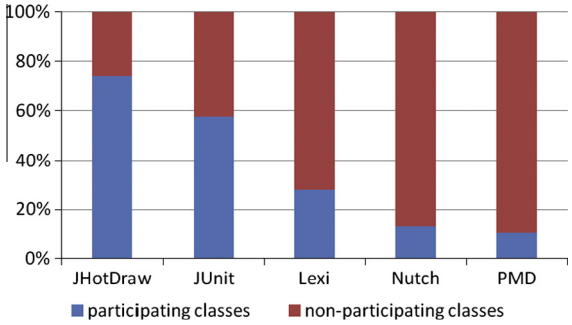

Fig. 1. Percentages of participant to non-participant classes.

Table 3
Number of design motifs' instances in the subject systems.

Category	Design motif	JHotDraw	JUnit	Lexi	Nutch	PMD
Creational design motifs	Abs. Factory					
	Builder			1		2
	Factory Method	3				3
	Prototype	2				
	Singleton	2	2	2	1	
Structural design motifs	Adapter	1			2	1
	Bridge				2	
	Composite	1	1			2
	Decorator	1	1			
	Facade					
	Flyweight					
	Proxy					1
Behavioral design motifs	Chain of Resp.					
	Command	1			2	
	Interpreter					
	Iterator		1		1	1
	Mediator					
	Memento				2	
	Observer	2	3	2		2
	State	2				
	Strategy	4			2	
	Template	2			3	1
	Visitor					1
Total # of instances		21	8	5	15	14

set of design motifs. Secondly, their precision and recall are either unknown or unsatisfactory. Finally, they do not detect all the roles in each design motif. Table 2 provides a comparison of design motifs detection tools in terms of the number of detectable design motifs, precision, and recall if known. Consequently, we searched for another source of design motifs data. As a result, we identified the P-Mart repository, which is a reliable source of design motifs data that has been used in several works [4,7,11,12,21]. It is considered a benchmark and golden standard in the literature [7]. The P-Mart repository contains design motifs data for nine systems. We performed our analysis on five of them because we did not find the faults data for the other four systems.

We collected faults data from the development CVS files (Concurrent Versions System) associated with each subject system. Our subject systems are hosted with their CVS files in SourceForge. CVS files contain the commits entries associated with each class file in the systems. Unlike change logs that are automatically generated from the CVS commits, release notes are produced manually to include only important/main changes and thus usually incomplete and inaccurate. We found that the release notes for the subject systems are incomplete. For example, the number of reported bugs in the release notes of the PMD subject system is only about 80, whereas the number of reported bugs in the CVS files of the same system is about 580. The same conclusion is also supported with the findings of Chen et al. [10]. So, we used CVS commits instead of release notes. To collect faults data, we examined each commit for each class in each subject system looking for the following keywords: fault, bug, error, defect, flaw, and issue. We excluded few cases where these words are mentioned for other purposes, such as clarification. After counting the number of faults in each class, we calculated the fault density of each class by dividing the number of faults by KLOC (Thousands Lines of Code) in each class. Only logical lines in method bodies were counted, while comments and blank lines were excluded.

We considered only main (top-level) classes. Nested classes (inner classes and static member classes) were treated as contents of the enclosing top-level classes. They were not treated as individual observations (data points) because none of them play any role in design motifs in the analyzed subject systems [22].

Table 3 reports the number of instances of each design motif in each one of the subject systems and Table 4 provides descriptive statistics for each subject system. The number of design motifs, the percentage of the participating classes in design motifs, and the percentage of faulty classes (i.e., classes that have at least one fault) are different from one system to the other. The numbers of design motifs range from 5 to 21 in the different systems and the percentages of the participating classes range from 10.6% to 74%, as shown in Fig. 1. The numbers and percentages of faulty classes range from 9 (11.5%) to 233 (52.2%) in the subject classes, as shown in Fig. 2. Table 5 provides descriptive statistics of fault density in

Table 4
Descriptive statistics of the subject systems.

Systems	# Of classes	Total LOC	# Of faulty classes	# Of participating classes in design motifs
JHotDraw	155	8891	45 (29.0%)	115 (74.0%)
JUnit	78	4773	9 (11.5%)	45 (57.6%)
Lexi	24	7045	11 (44.0%)	7 (28.0%)
Nutch	165	23,507	74 (44.8%)	22 (13.3%)
PMD	446	41,486	233 (52.2%)	49 (10.6%)
All systems	868	85,702	372 (42.8%)	238 (27.4%)

freepaper.me FREE paper

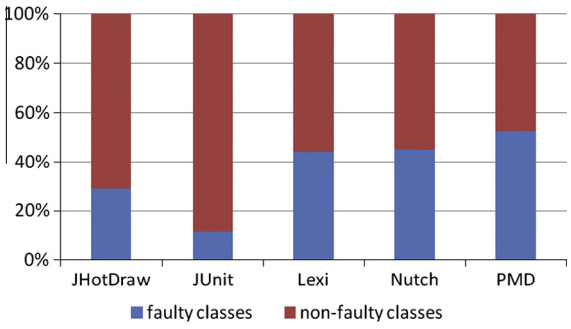

Fig. 2. Percentages of faulty to non-faulty classes.

each subject systems, and Fig. 3 shows box-plots of fault density in each subject system.

3.4. Statistical tests

We used the Mann–Whitney U test to measure the difference between two groups and the Kruskal–Wallis test to measure the difference among more than two groups.

3.4.1. Mann–Whitney test

The Mann–Whitney U test is a non-parametric test used to compare the differences between two independent groups [35]. We used this test to evaluate the differences between the different groups. For example, we used it to compare the difference in fault density between classes that participate in design motifs and non-participant classes. As other example, we used this test to evaluate the difference in fault density between classes that participate in structural design motifs and classes that participate in creational design motifs. There are four assumptions that must be held to perform this test. These assumptions are as follows:

1. Dependent variables should be either ordinal or continuous. In our case, fault density is a continuous variable.
2. Independent variable should consist of two categorical groups. In our case, the values of the class participation in design motifs are either 0 or 1; where 0 refers to the first group and 1 refers to the second group.
3. There should be an independence of observations. In our case, there is no relationship between the different observations so one observation does not affect another observation (i.e., if a class is a participant in one design motif does not imply or require to be or not to be a participant in another motif).
4. The two independent variables (i.e., groups) should not be normally distributed. To test the normality of the data, we performed two tests: Kolmogorov–Smirnov and Shapiro–Wilk tests. These tests are performed on fault density data of participant versus non-participant classes when all systems are combined together. Table 6 provides the results of these tests. The assumption in evaluating the normality is "the data are not normally distributed" and it was found that the *p*-value associated with evaluating the normality is less than 0.05.

Table 5
Descriptive statistics of fault density in each subject system.

	JHotDraw	JUnit	Lexi	Nutch	PMD	All systems
Mean	13.5	6.6	5.1	10.1	34.1	22.6
Minimum	0	0	0	0	0	0
Maximum	285.7	375.0	31.3	125.0	800.0	800.0
Std. dev.	36.1	43.0	8.4	18.9	62.5	51.2

Table 6
Tests of normality.

Data set	Groups	Kolmogorov–Smirnov (Significance)	Shapiro–Wilk (Significance)
Fault density	Non-participant	<0.001	<0.001
	Participant	<0.001	<0.001

3.4.2. Kruskal–Wallis test

The Kruskal–Wallis test is a non-parametric test used to compare the differences between two or more independent groups [32]. We used this test to evaluate the differences among the different roles in each design motif. Because we are comparing more than two groups using this test, we used Bonferroni method for correcting the obtained *p*-values. There are two assumptions that must be held to perform this test. These assumptions are as follows:

1. Dependent variables should be either ordinal or continuous. In our case, fault density is a continuous variable.
2. Independent variable should consist of more than two categorical groups. In our case, this variable consists of more than two groups (roles) in all of the addressed design motifs except for one, which is Singleton that consists of one role only. All other motifs consist of more than two roles.

4. Empirical study results

We conducted the quantitative analysis of fault density in design motifs at four levels (design level, category level, motif level and role level) as follows.

4.1. Design level

At the design level, we evaluated the difference in fault density between classes that participate in any design motif and classes that do not participate in any design motif (i.e., participant vs. non-participant). This comparison gives a general insight into the association between design motifs and fault density. The obtained results, from the Mann–Whitney test, show only two significant (<0.05) *p*-values in the case of Lexi and when we combine all systems, as provided in Table 7. The test is also significant in JHotDraw but at 0.1 level of significance.

Fig. 4 shows box-plots of the fault density of design motifs' participant vs. non-participant classes in each subject system. These plots are interpreted by the location of the median, and the location and size of the box and whiskers (i.e., the lines above and

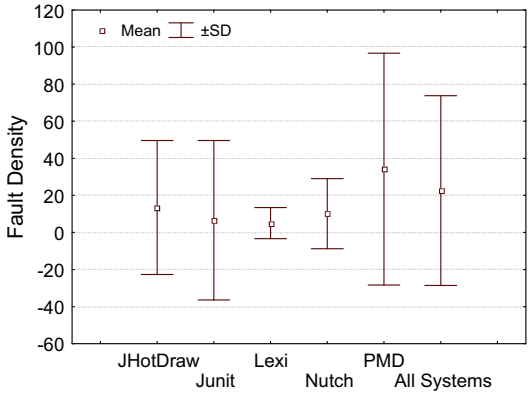

Fig. 3. Box-plots of fault density in each subject system.

freepaper.me FREE paper دانلود کننده مقالات علمی

Table 7
Comparison evaluation results (p-values) of design motifs' participant vs. non-participant classes in each subject system.

	Participant vs. non-participant
JHotDraw	0.057677
JUnit	0.386525
Lexi	**0.011515**
Nutch	0.788716
PMD	0.971235
All systems	**0.000006**

Bold values are significant at the 0.05 level (95% confidence level).

below the box). The group that is less dense in faults will have its median point in relatively lower position, its box should be narrower, and its whiskers should be smaller than the other group. In case of JHotDraw, PMD, and when we combine all systems, we found that participant classes are less fault-dense than non-participant classes. In case of Lexi, we found an opposite result, i.e., participant classes are more fault-dense than non-participant classes. In case of JUnit, the two groups seem similar. In case of Nutch, the participating classes group has higher median but narrower boxes and smaller whisker than the non-participating classes group. Based on these results, we conclude that there is no clear tendency for the difference in fault density between participant and non-participant classes in design motifs. We therefore cannot accept or reject the null Hypothesis 1.

4.2. Category level

The objective of this section is to evaluate the differences in fault density of classes that participate in the different categories of design motifs, i.e., creational, structural and behavioral. We identify six pairs of categories (creational vs. non-participant,

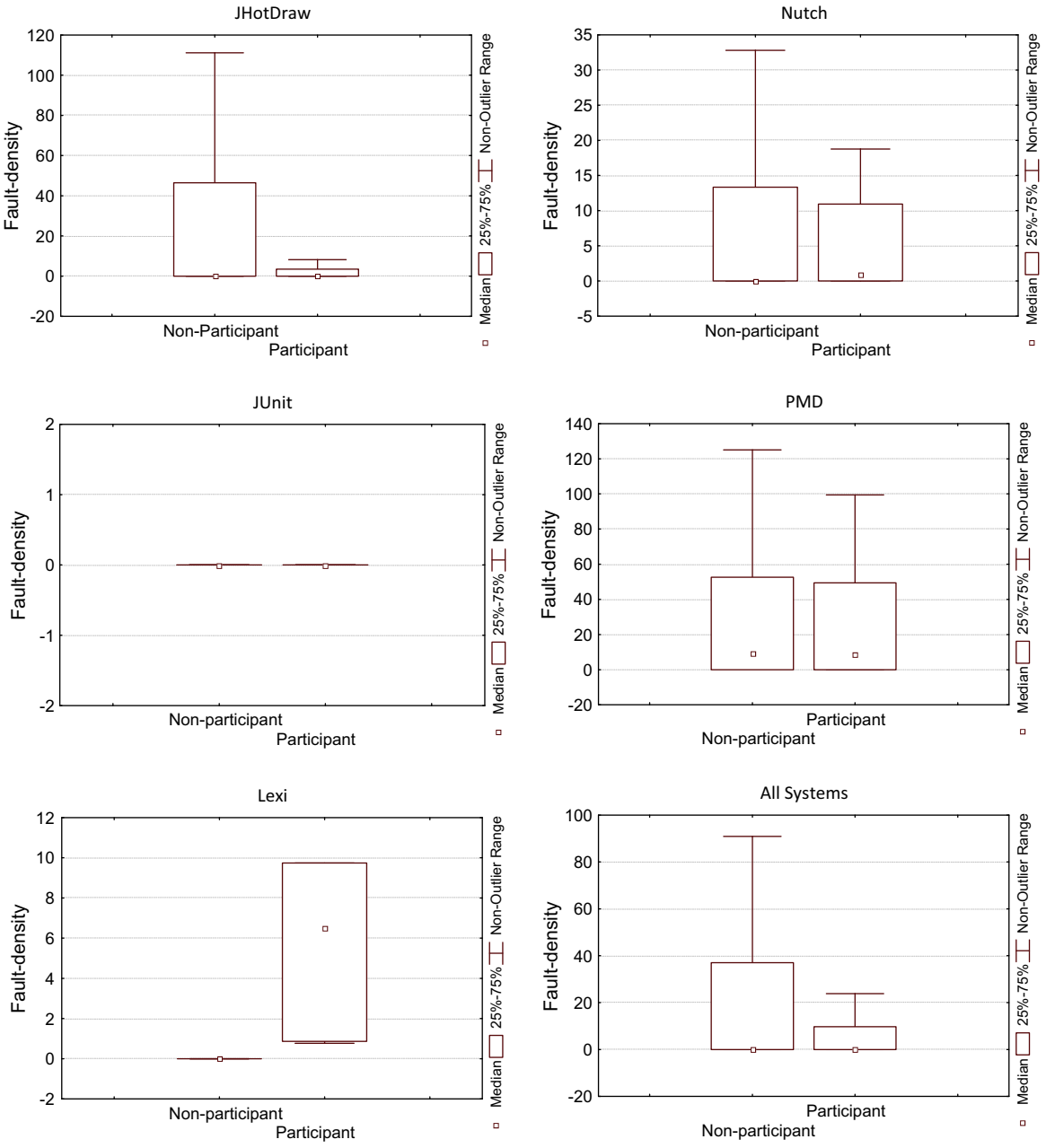

Fig. 4. Box-plots of the fault density of participant and non-participant classes in each subject system.

freepaper.me FREE paper

Table 8
Comparison evaluation results (p-values) of the different categories of design motifs in each subject system.

	Creational	Structural	Behavioral
JHotDraw			
Non-participant	**0.040954**	0.074557	0.190715
Creational	–	0.623715	**0.010846**
Structural		–	0.089498
JUnit			
Non-participant	0.129812	0.094472	1.000000
Creational	–	**0.005415**	0.204005
Structural		–	0.116530
Lexi			
Non-participant	**0.011515**	–	**0.036336**
Creational	–	–	1.000000
Structural		–	–
Nutch			
Non-participant	–	0.309721	0.113882
Creational	–	–	–
Structural		–	**0.024861**
PMD			
Non-participant	0.201336	0.774087	0.091711
Creational	–	0.476195	**0.041217**
Structural		–	0.299755
All systems			
Non-Participant	0.082337	**0.000000**	**0.000480**
Creational	–	**0.000135**	0.194804
Structural		–	**0.005921**

Bold values are significant at the 0.05 level (95% confidence level).

structural vs. non-participant, behavioral vs. non-participant, creational vs. structural, creational vs. behavioral, and structural vs. behavioral) for evaluation in each subject system and when we combine all systems. In the evaluation cases that involve creational design motifs, we ignored Nutch because only one of its classes participates in creational design motifs. We also ignored Lexi in the cases that involve structural design motifs because it has no structural design motifs. Table 8 summarizes the evaluation results of these pairs by providing the p-values of the Mann–Whitney tests where significant values are in boldface. Based on these results, we reject the null Hypothesis 2 and accept its alternative hypothesis.

4.2.1. Creational vs. non-participant

We obtained two significant Mann–Whitney test p-values in different directions in measuring the fault density of non-participant classes versus classes that participate in creational design motifs. These values are associated with JHotDraw and Lexi, as shown in Table 8. Fig. 5 shows that non-participant classes are more fault-dense than classes that participate in creational design motifs in JHotDraw and less fault-dense than classes that participate in creational design motifs in Lexi. The tendency for the difference in fault density for non-participant classes versus creational classes is not clear. In addition to the different directions for the significant p-values, we found that non-participant classes in PMD and JUnit are less fault-dense than classes that participate in creational design motifs and more fault-dense when we combine all systems.

4.2.2. Structural vs. non-participant

We obtained one significant Mann–Whitney test p-value in evaluating the fault density of non-participant classes versus classes that participate in structural design motifs, as shown in Table 8, when we combine all systems. Fig. 5 shows that non-participant classes are more fault-dense than classes that participate in structural design motifs. For JHotDraw and JUnit, the differences are not significant at the 0.05 level. However, they are significant at the 0.1 level, as shown in Table 8. They also support the results obtained when we combine all systems. The same thing can be said about the tendency for the difference in fault density in the cases of Nutch and PMD, even though the p-values associated with them are not significant. Therefore, non-participant classes tend to be more fault-dense than classes that participate in structural design motifs.

4.2.3. Behavioral vs. non-participant

The evaluation of the difference in fault density of non-participant classes compared to classes that participate in behavioral design motifs resulted in two significant Mann–Whitney test p-values in two different directions. The first value is associated with Lexi and the second value was obtained when we combine all systems, as shown in Table 8. Fig. 5 shows that non-participant are less fault-dense than classes that participate in behavioral design motifs in Lexi and more fault-dense than classes that participate in behavioral design motifs when we combine all systems. It is difficult to decide which group tends to be more fault-dense: non-participant classes or classes that participate in behavioral design motifs because the conclusion drawn from JHotDraw, PMD, JUnit, and when we combine all systems is in one direction and the conclusion drawn from Lexi and Nutch is in the other direction, as shown in Fig. 5.

4.2.4. Creational vs. structural

We obtained only two significant p-values as a result of comparing the fault density of classes that participate in creational design motifs and classes that participate in structural design motifs. These values were obtained in JUnit and when we combine all systems, as shown in Fig. 5. In these cases, classes that participate in creational design motifs are more fault-dense than classes that participate in structural design motifs. This finding is supported by JHotDraw and PMD. Although the p-values associated with these two systems are not significant, they provide support for the results obtained in JUnit and when we combine all systems. Therefore, classes that participate in creational design motifs tend to be more fault-dense than classes that participate in structural design motifs.

4.2.5. Creational vs. behavioral

We found two significant p-values in two different directions in evaluating the difference in fault density between classes that participate in creational design motifs and classes that participate in behavioral design motifs, as shown in Table 8. These values are associated with JHotDraw and PMD. In JHotDraw, classes that participate in creational design motifs are less fault-dense than classes that participate in behavioral design motifs, as shown in Fig. 5. In PMD, classes that participate in creational design motifs are more fault-dense than classes that participate in behavioral design motifs. It is difficult to decide which group tends to be more fault-dense: classes that participate in creational design motifs or classes that participate in behavioral design motifs because the differences in JHotDraw and Lexi are in one direction and in PMD, JUnit, and when we combine all systems are in the other direction.

4.2.6. Structural vs. behavioral

We obtained two significant p-values in the same direction in evaluating the difference in fault density between classes that participate in structural design motifs and classes that participate in behavioral design motifs, as shown in Table 8. These values are associated with Nutch and when we combine all systems. In both cases, classes that participate in structural design motifs are less fault-dense than classes that participate in behavioral design motifs, as shown in Fig. 5. The p-value associated with JHotDraw is insignificant though it provides further support for the same conclusion. In PMD, the obtained result is in the opposite direction

freepaper.me FREE paper

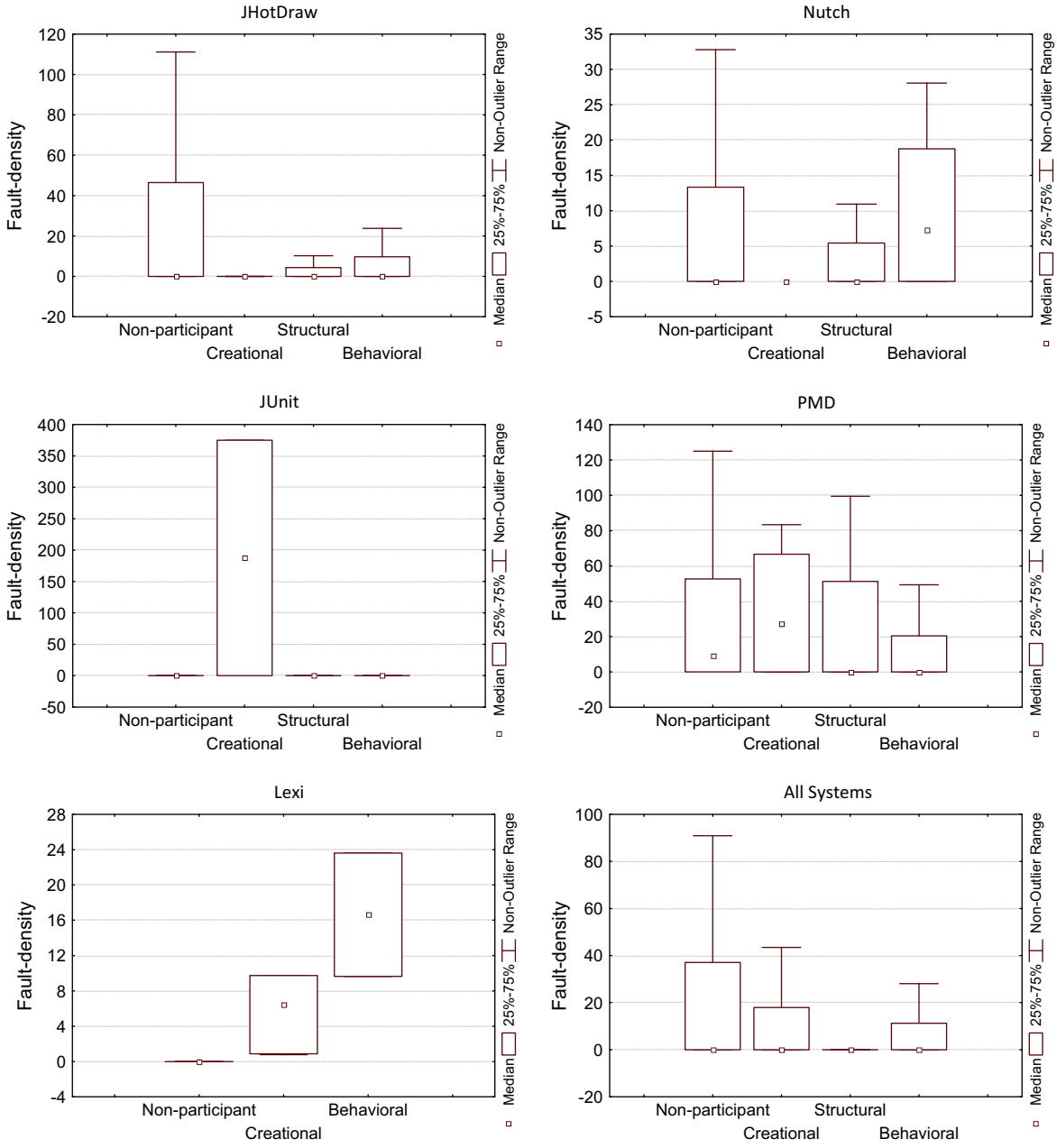

Fig. 5. Box-plots of the fault density in the different categories of design motifs in each subject system.

but it is not significant, as shown in Fig. 5. Moreover, in JUnit, we did not observe a difference in fault density. We conclude that classes that participate in structural design motifs tend in general to be less fault-dense than classes that participate in behavioral design motifs.

4.3. Motif and role levels

In measuring and comparing the fault density of each design motif, we evaluate:

- The difference in fault density of classes that participate in a design motif and classes that do not participate in that design motif using the Mann–Whitney test.
- The overall difference among the different roles in each design motif using the Kruskal–Wallis test.
- The differences between each pair of roles in a design motif if we find differences among the different roles.

We used the data from all systems combined to measure and compare the fault density of each design motif and its roles. We did not conduct this analysis for each subject system because the set of design motifs in each system is different and, thus, we cannot compare results across systems. Although the 'client' role is not one of the application roles, we consider it because it is listed as one of the patterns' participants in the GoF's book [15].

In our dataset, 46.2% of classes that participate in design motifs play more than one role. We accordingly investigated the possible correlation between the number of roles played by classes and their fault density. The results of a Spearman correlation analysis indicate very weak correlation (coefficient: −0.0753 and p-value: 0.2472). We therefore did not perform further analysis based on

the number of roles. Khomh et al. [28], however, found that the number of roles do impact the number and frequencies of changes. There are several reasons that may explain why their observations are different from ours. First of all, they investigated the impact on change-proneness, but not on fault density. Second, they selected six design motifs and considered only their main roles. Third, they analyzed different systems. Finally, they identified design motifs using automatic detection technique, DeMIMA, which produces false positives.

4.3.1. Creational design motifs

Table 9 provides summary of the *p*-values of the tests of significance differences in fault density that we conducted for each creational design motif and its roles. Significant *p*-values are in boldface. The "overall roles comparison" in this table refers to the result of the Kruskal–Wallis test that measures the significance difference among the different roles in the corresponding design motif. We report the observations for each creational design motif next.

- **Builder.** The Builder motif classes are significantly more fault-dense than the non-Builder motif classes. This result is also supported by Fig. 6(a) that shows box-plots of the fault density in Builder and non-Builder classes. Although there are some differences in the fault density among the participating classes in the different roles in the Builder design motif, as shown in Fig. 6(b), these differences are not significant at the 0.05 level but they are significant at the 0.1 level. The Director role, which is responsible of notifying the Builder whenever a part of a product should be built, is more dense in faults than other roles, and thus should be implemented carefully. For the Builder design motif, we reject the null Hypothesis 3 and accept its alternative hypothesis. We also accept the null Hypothesis 4.
- **Factory Method.** There is a significant difference in the fault density between the Factory Method classes and the non-Factory Method classes. As shown in Fig. 7(a), the Factory Method classes are less fault-dense than the non-Factory Method classes. In addition, there are significant differences in the fault density among classes that participate in the different roles in the Factory Method design motif. The only pair of roles that shows a significant difference is: Concrete-Product vs. non-participant, where classes that participate in the Concrete-Product role are less fault-dense than non-participant classes, as shown in Fig. 7(b). For the Factory Method design motif, we reject the null Hypothesis 3 and accept its alternative hypothesis. We also reject the null Hypothesis 4 and accept its alternative hypothesis.
- **Prototype.** There is no significant difference in the fault density between the Prototype classes and the non-Prototype classes. Moreover, there are no significant differences in the fault density among classes that participate in the different roles in the Prototype design motif. For the Prototype design motif, we accept the null Hypothesis 3 and also that of Hypothesis 4.

Table 9
Comparison evaluation results of creational design motifs and their roles.

Design motif	Pairs	*p*-value
Builder	Builder classes vs. Non-Builder classes	**0.043**
	Overall roles comparison	0.074
Factory Method	Factory Method Classes vs. Non-Factory classes	**0.037**
	Overall roles comparison	**0.034**
	Concrete-Product vs. Non-Participant	**0.003**
Prototype	Prototype classes vs. Non-Prototype classes	0.221
	Overall roles comparison	0.665
Singleton	Singleton classes vs. Non-Singleton classes	0.773

Bold values are significant at the 0.05 level (95% confidence level).

- **Singleton.** There is no significant difference in the fault density between the Singleton classes and the non-Singleton classes. Roles evaluation is not applicable for this design motif because it has only one role. For the Singleton design motif, we accept the null Hypothesis 3.

4.3.2. Structural design motifs

Table 10 summarizes the *p*-values of the tests of significance differences in fault density for each structural design motif and its roles, where significant *p*-values are in boldface. We report the observations for each structural design motif next.

- **Adapter.** The Adapter motif classes are significantly less fault-dense than the non-Adapter motif classes. Fig. 8 shows box-plots of the fault density in the Adapter design motif and its roles. At the role level, there are two pairs of roles that have significant differences in fault density between themselves. The first pair is Adapter vs. non-participant, where classes that participate in the Adapter role are less fault-dense than non-participant classes. The second pair is Adapter vs. Client, where classes that participate in the Adapter role are less fault-dense than the Client role. For the Adapter design motif, we reject the null Hypothesis 3 and accept its alternative hypothesis. We also reject the null Hypothesis 4 and accept its alternative hypothesis.
- **Bridge.** There is no significant difference in the fault density between the Bridge classes and the non-Bridge classes. In addition, there are no significant differences in the fault density among classes that participate in the different roles in the Bridge design motif. For the Bridge design motif, we accept the null Hypothesis 3 and also that of Hypothesis 4.
- **Composite.** There is a significant difference in the fault density between the Composite motif classes and the non-Composite motif classes. As shown in Fig. 9(a), the Composite motif classes are less fault-dense than the non-Composite motif classes. In addition, there are significant differences in the fault density among classes that participate in the different roles in the Composite design motif. The only pair of roles that shows a significant difference is: Leaf vs. non-participant, where classes that participate in the Leaf role are less fault-dense than non-participant classes, as shown in Fig. 9(b). For the Composite design motif, we reject the null Hypothesis 3 and accept its alternative hypothesis. We also reject the null Hypothesis 4 and accept its alternative hypothesis.
- **Decorator.** There is a significant difference in the fault density between the Decorator motif classes and the non-Decorator motif classes. As shown in Fig. 10(a), the Decorator motif classes are less fault-dense than the non-Decorator motif classes. At the role level, there are two pairs of roles that have significant differences in fault density between themselves. The first pair is Concrete-Decorator vs. non-participant, where classes that participate in the Concrete-Decorator role are less fault-dense than non-participant classes, as shown in Fig. 10(b). The second pair is Concrete-Component vs. non-participant, where classes that participate in the Concrete-Component role are less fault-dense than non-participant classes, as shown in Fig. 10(b). For the Decorator design motif, we reject the null Hypothesis 3 and accept its alternative hypothesis. We also reject the null Hypothesis 4 and accept its alternative hypothesis.
- **Proxy.** There is no significant difference in the fault density between the Proxy classes and the non-Proxy classes. Moreover, there are no significant differences in the fault density among classes that participate in the different roles in the Proxy design motif. For the Proxy design motif, we accept the null Hypothesis 3 and also that of Hypothesis 4.

freepaper.me FREE paper

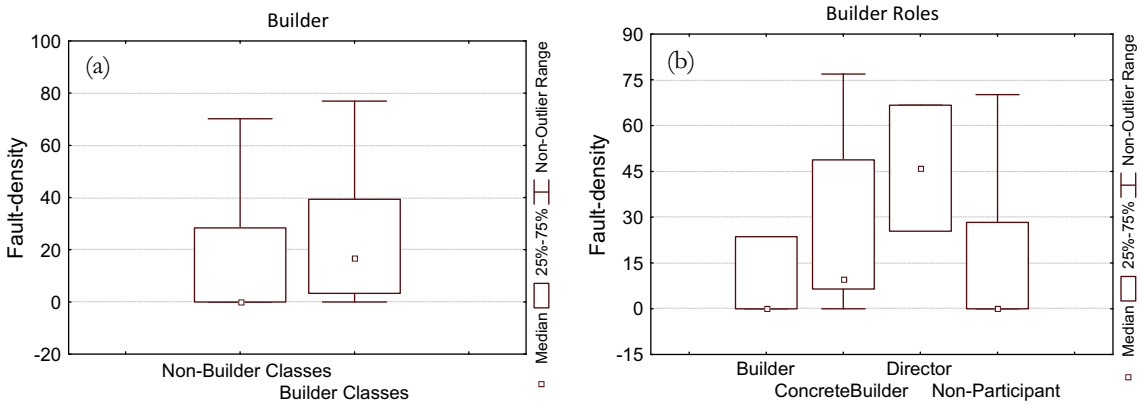

Fig. 6. Box-plots of the fault density in the Builder design motif and its roles.

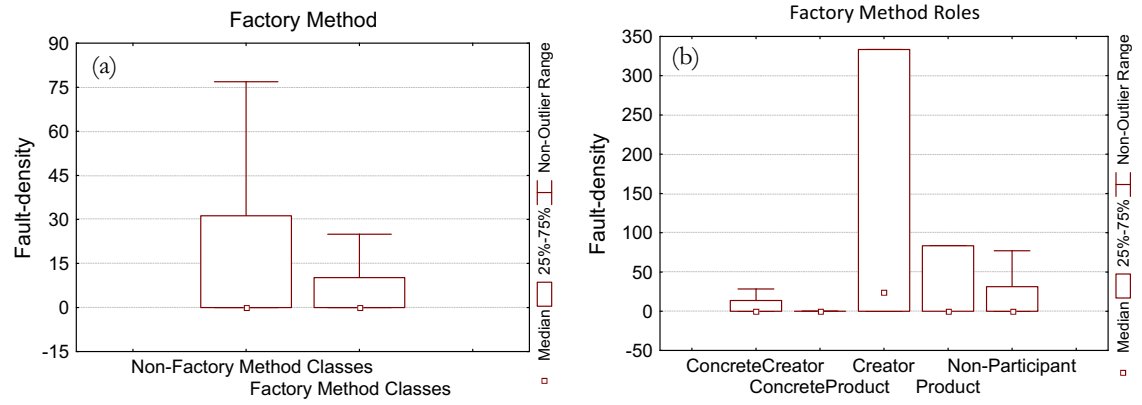

Fig. 7. Box-plots of the fault density in the Factory Method design motif and its roles.

Table 10
Comparison evaluation results of structural design motifs and their roles.

Design motif	Pairs	p-value
Adapter	Adapter classes vs. Non-Adapter classes	**0.001**
	Overall roles comparison	**0.003**
	Adapter vs. Non-participant	**0.000**
	Adapter vs. Client	**0.019**
Bridge	Bridge classes vs. Non-Bridge classes	0.400
	Overall roles comparison	0.758
Composite	Composite classes vs. Non-Composite classes	**0.000**
	Overall roles comparison	**0.001**
	Leaf vs. Non-participant	**0.000**
Decorator	Decorator classes vs. Non-Decorator classes	**0.000**
	Overall roles comparison	**0.001**
	Concrete-Decorator vs. Non-participant	**0.036**
	Concrete-Component vs. Non-participant	**0.000**
Proxy	Proxy classes vs. Non-Proxy classes	0.153
	Overall roles comparison	0.563

Bold values are significant at the 0.05 level (95% confidence level).

4.3.3. Behavioral design motifs

Table 11 provides summary of the *p*-values of the tests of significance differences in fault density for each behavioral design motif and its roles, where significant *p*-values are in boldface. We found no significant differences in fault density in any behavioral design motif neither at the motif level nor at the role level. Therefore, for each of behavioral design motifs, we accept the null Hypothesis 3 and also that of Hypothesis 4.

5. Summary and discussion

In this section, we summarize and discuss the obtained results.

5.1. Design level

Table 12 is a summary for the results described in Section 4.1. There are no commonalities on the association between design motifs and fault density. To deduce a general summary, we allow, at most, one insignificant case anomaly from the derived conclusion and we ask for, at least, one significant case that supports the derived conclusion and the rest of the cases must support the derived conclusion even if they are not significant. In Table 12, in four cases, participant classes are more fault-dense "(>)" than non-participant classes. One of these cases is associated with significant "(S)" *p*-value. In the other three cases, the associated *p*-values are not significant "(N)". Also, in two cases, participant classes are less fault-dense "(<)" than non-participant classes. One of these cases is associated with significant *p*-value "(S)". In the other case, the associated *p*-value is not significant "(N)". The number between the parentheses "(6)" refers to the number of applicable cases.

Based on the above results, the answer to RQ1 is that there is no clear tendency for the difference in fault density between participant and non-participant classes in design motifs. The absence of commonalities among the different cases might be due to the intentional uses of design motifs and the level of experience in implementing them. It is mentioned in the documentation of two systems (JHotDraw and JUnit) that developers used design motifs. It is however unknown if developers used design motifs intentionally in the other subject systems. Among other possible reasons for the absence of commonalities are the sizes and complexities of the subject systems and the numbers and types of design motifs instances in them. A controlled experiment may be conducted in the future to study possible cause-and-effect relationships.

freepaper.me FREE paper

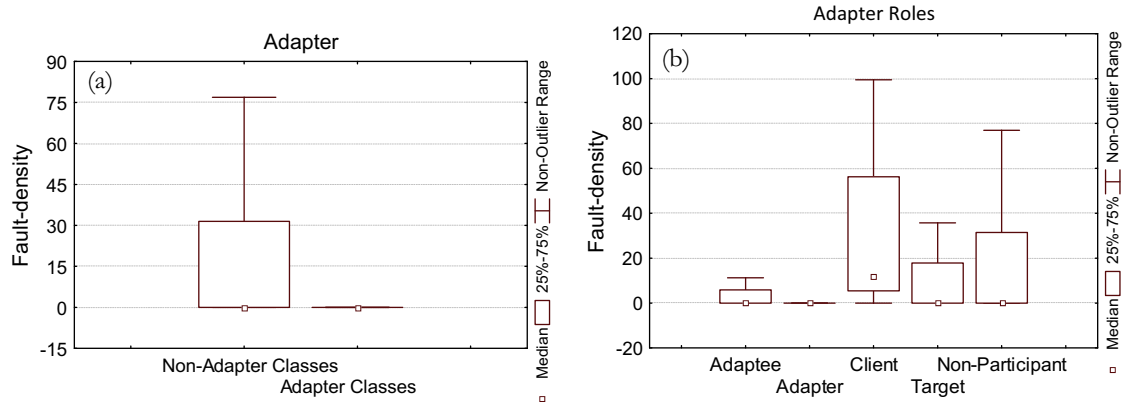

Fig. 8. Box-plots of the fault density in the Adapter design motif and its roles.

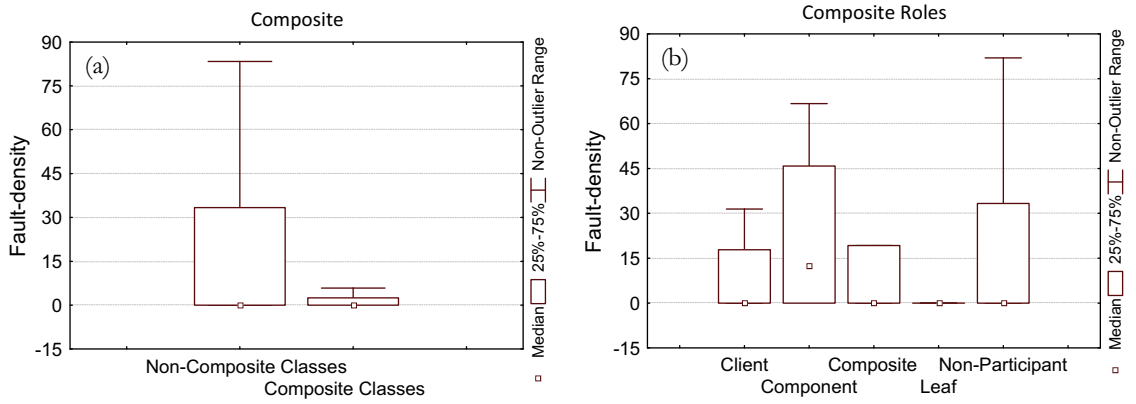

Fig. 9. Box-plots of the fault density in the Composite design motif and its roles.

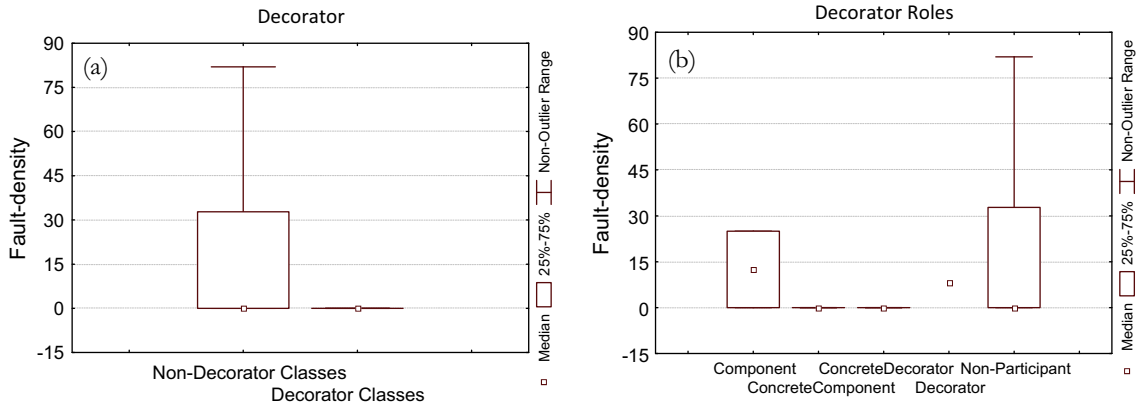

Fig. 10. Box-plots of the fault density in the Decorator design motif and its roles.

5.2. Category level

We evaluated the difference in fault density of the different categories of design motifs to find if there is an association between the different categories of design motifs and the fault density and consequently to answer RQ2. Tables 13 and 14 provide brief and detailed summaries, respectively, of the differences among the different categories of design motifs on fault density. The number of applicable cases is different from one pair to the other. It ranges from four to six because the number of classes that participate in creational category of Nutch is one, and because Lexi has no structural design motifs.

We found that structural design motifs are less fault-dense than classes that participate in creational and behavioral design motifs and also less fault-dense than non-participant classes. This observation suggests that the adoption of structural design motifs might result in more reliable software because these design motifs tend to be less fault-dense. The reason might be due to that structural design motifs are related to the structure of the system, which may be less prone to change [28,40]. In addition, we think the idea of these design motifs are easy to grasp and consequently easy to apply in software design. Based on our teaching experience, we found that students find it easier to understand structural design motifs compared to behavioral design motifs.

freepaper.me FREE paper

Table 11
Comparison evaluation results of behavioral design motifs and their roles.

Design motif	Pairs	*p*-value
Command	Command classes vs. Non-Command classes	0.338
	Overall roles comparison	0.671
Iterator	Iterator classes vs. Non-Iterator classes	0.785
	Overall roles comparison	0.549
Memento	Memento classes vs. Non-Memento classes	0.830
	Overall roles comparison	0.829
Observer	Observer classes vs. Non-Observer classes	0.149
	Overall roles comparison	0.235
State	State classes vs. Non-State classes	0.602
	Overall roles comparison	0.238
Strategy	Strategy classes vs. Non-Strategy classes	0.094
	Overall roles comparison	0.182
Template Method	Template Method classes vs. Non-Template classes	0.575
	Overall roles comparison	0.725
Visitor	Visitor classes vs. Non-Visitor classes	0.994
	Overall roles comparison	0.407

5.3. Motif level

To answer RQ3, we evaluated the difference in fault density between classes that participate in each design motif and classes that do not participate in that motif. We found that, in all of structural design motifs that show significant difference (i.e., Adapter, Composite and Decorator), classes that participate in the design motifs are less fault-dense than non-participant classes, as shown in Table 15. This observation explains the obtained results in evaluating the different categories in the previous section. Also, we found that behavioral design motifs have no association with fault density. Furthermore, only five design motifs out of the 17 examined design motifs have significant associations with fault density. The Builder design motif shows a positive association with fault density whilst the Factory Method, Adapter, Composite, and Decorator design motifs show negative associations with fault density.

The observed associations between the Builder, Factory Method, Adapter, Composite, and Decorator design motifs and fault density can be explained as follows. The positive association between the Builder design motif and fault density might be due to that the Builder design pattern is one of the unfamiliar and least favored patterns based on a survey of experienced user perceptions about software DPs by Zhang and Budgen [49]. The negative association between the Factory Method design motif and fault density could be because it has simple structure and relatively small amounts of code that is loosely coupled to the rest of the system [47]. The negative association between the Adapter design motif and fault density might be due to its structure is simple and it is straightforward to implement [15]. The negative association between the Composite design motif and fault density could be because the use of Composite design pattern should not be problematic according to a summary of pervious qualitative assessments [50]. Extension to a Composite design motif should not introduce faults because newly defined Composite and Leaf subclasses work automatically with its existing structure [15]. Furthermore, the Composite design pattern is one of the patterns that are very well known, liked, and highly valued according to Zhang and Budgen's survey [49]. The negative association between the Decorator design motif and fault density was surprising because the call graph of a Decorator pattern structure is difficult to understand and maintain [3,47]. However, in this study, there is only one instance of the Decorator in JHowDraw and another in JUnit that seem to be well applied.

Although previous work [3,17,47] addressed other attributes of faults, such as fault-proneness and frequency, we generally compare their results to ours with respect to the relationship between design motifs and faults. In comparing our results to the results obtained by other studies, we find some similarities and some differences. Table 16 compares our observed associations between design motifs and fault density with previous work, where an empty cell means that the corresponding design motif was not examined by the corresponding study, and the '+', '−', and '×' symbols indicate positive association, negative association, and no association, respectively. Some of the contradictions and diversities in Table 16 can be explained as follows. In case of the Adapter design motif, there is extensive use of it (many instances) in the analyzed systems in [3,17], but there are few instances of it in our study. The extensive use of the Adapter design motif might insert faults in the system either because the developers of the system are often not familiar with the adapted code or the faults of the adapted code are added to the faults of the target system [3]. In case of the Decorator design motif, there is no enough instances of it in the analyzed system in [47] to yield statistically significant results. Although, in our study, there are also few instances of the Decorator design motif, they are from JHotDraw and JUnit where design motifs were intentionally used in these two systems, and thus expected to be of good quality. In case of the Observer design motif, Vokac [47] explained the high fault frequency in this design motif by its complex structure and usage in situations with coupling between multiple, nontrivial classes. However, Ampatzoglou et al. [3] argued that because the Observer design motif has complex structure, it is probably applied by more experienced developers, leading to less faults.

5.4. Role level

To answer RQ4, we summarized the differences in fault density among the different roles in each design motif. Table 17 shows that only a few design motifs show significant difference in fault density among their roles. Only one pair of roles shows significant difference: Adapter vs. Client. In this pair, classes that participate in the Client role are more fault-dense than classes that participate in the Adapter role. We think this observation might be due to Client role classes being more involved in implementing complex functionality than the Adapter role classes. The Adapter role classes are of less complexity because that they are only responsible for adapting other classes' methods to be used by the client. Other than this pair, all the other differences, in Table 17, are associated with evaluating the differences between non-participant classes and some design motif roles.

Table 12
Detailed summary for the results at the design level.

First group	Fault density				Second group
Direction	>		<		
Significance	S	N	S	N	
Participants (6)	1	3	1	1	Non-participants

S: # of significant cases and N: # of insignificant cases; (#) is the number of applicable cases.

Table 13
Summary of the differences among the different categories of design motifs on fault density.

First group	Fault density	Second group
Creational	–	Non-participant
Structural	<	Non-participant
Behavioral	–	Non-participant
Creational	>	Structural
Structural	<	Behavioral
Behavioral	–	Creational

freepaper.me FREE paper

Table 14
Detailed summary for the results at the category level.

First group	Fault density				Second group
Direction	>		<		
Significance	S	N	S	N	
Creational (5)	1	2	1	1	Non-participant
Structural (5)	0	0	1	4	Non-participant
Behavioral (6)	1	1	1	3	Non-participant
Creational (4)	2	2	0	0	Structural
Creational (5)	1	2	1	1	Behavioral
*Structural (5)	0	1	2	1	Behavioral

S: # of significant cases and N: # of insignificant cases; (#) is the number of applicable cases; * means that the absent case in evaluating the fault density has no differences between structural and behavioral groups.

Table 15
Summary of the differences among the different design motifs on fault density.

Design motifs category	First group	Fault density	Second group
Creational	Builder classes	>	Non-Builder classes
	Factory Meth. classes	<	Non-Factory Meth. classes
	Prototype classes	–	Non-Prototype classes
	Singleton classes	–	Non-Singleton classes
Structural	Adapter classes	<	Non-Adapter classes
	Bridge classes	–	Non-Bridge classes
	Composite classes	<	Non-Composite classes
	Decorator classes	<	Non-Decorator classes
	Proxy classes	–	Non-Proxy classes
Behavioral	Command classes	–	Non-Command classes
	Iterator classes	–	Non-Iterator classes
	Memento classes	–	Non-Memento classes
	Observer classes	–	Non-Observer classes
	State classes	–	Non-Participant classes
	Strategy classes	–	Non-Strategy classes
	Template Meth. classes	–	Non-Template Meth. classes
	Visitor classes	–	Non-Visitor classes

6. Threats to validity

6.1. Construct validity

Construct validity is concerned with the measures used in the evaluation. Faults data can be a threat to the construct validity of this study. To alleviate this risk, we have collected the faults data from the CVS files and not the release notes, and we have examined every commit. However, there might be other unreported faults, but we believe such cases (if any) to be very limited and thus do not affect the results significantly because the subject systems are popular, open-source, and widely used systems and their bug tracking systems are very active.

P-Mart repository has been created using different sources: studies in the literature [8]; Ptidej (pattern trace identification, detection, and enhancement in Java) tool for identifying design motifs [20]; and validation assignments for students in undergraduate and graduate courses. These different sources alleviate the possibility of false positive and negative instances of design motifs in the P-Mart repository. Another threat to construct validity is whether 'client' role should be considered or not among participant classes in design motifs. Future work may exclude this role from analysis.

6.2. Internal validity

Internal validity is the degree to which the observed effects depend only on the intended experimental variables. One threat is the number of roles that a class might be playing in design motifs. We did not find a correlation between the number of roles played by the classes and their fault density. Another threat to the internal validity emerges from the developers' background. We do not know whether the developers are trained to work with DPs or not. However, we are not studying cause-and-effect relationship because we cannot control each variable that affect the relationships among the different groups. We are only trying to see if there is a significant association between the addressed variables or not.

Table 16
Summary of observed associations between design motifs and fault density: comparison with previous work.

Category	Design motif	Vokac [47]	Gatrell and Counsell [17]	Ampatzoglou et al. [3]	This paper
Creational design motifs	Abs. Factory			–	
	Builder		×		+
	Factory Method	–	×		–
	Prototype			–	×
	Singleton	+	+	×	×
Structural design motifs	Adapter		+	+	–
	Bridge				×
	Composite			–	–
	Decorator	×		×	–
	Facade				
	Flyweight				
	Proxy		×	–	×
Behavioral design motifs	Chain of Resp.				
	Command		×		×
	Interpreter				
	Iterator		×		×
	Mediator				
	Memento				×
	Observer	+		–	×
	State		×	–	×
	Strategy		×	–	×
	Template	×		+	×
	Visitor		×		×

freepaper.me FREE paper

Table 17
Summary of the difference among the different roles in design motifs on fault density.

Design motif	First group	Fault density	Second group
Factory Method	Concrete-Product	<	Non-participant
Adapter	Adapter	<	Non-participant
	Adapter	<	Client
Composite	Leaf	<	Non-participant
Decorator	Concrete-Decorator	<	Non-participant
	Concrete-Component	<	Non-participant

6.3. External validity

External validity is concerned with generalizability. The nature of the subject systems is a threat to the external validity of this study. All subject systems are open-source systems and developed using one programming language-Java. To generalize the obtained results in this study, we must investigate the design motifs further by including commercial systems and systems developed using other programming languages. However, this study can be considered as a step that can be strengthened later with more replications.

6.4. Conclusion validity

Conclusion validity is the degree to which the conclusions that are obtained about the relationships in the testing data are reasonable. We performed our analysis at the design level and the category level on six cases (i.e., five subject systems and when all of these systems are combined together). 70% of the classes in all systems are from PMD and Nutch; and 82% of faulty classes in all systems are from PMD and Nutch. However, only 30% of participant classes in design motifs in all systems are from PMD and Nutch. For this reason, the obtained results show that there is no consensus among the six cases. Moreover, two cases (Lexi and JUnit) have relatively few classes and instances of design motifs. To draw conclusions on the general tendencies for the differences in fault density, we allow only for one case anomaly at most, from the direction of the obtained conclusion, which is not associated with significant p-value. Also, there should be at least one case that shows significant difference in the direction of the obtained conclusion. The other cases should be in the same direction of the obtained conclusion. This approach is better and more restricted than the majority voting and less restricted than the consensus. Considering more cases or more systems may lead to different conclusions based on these criteria.

7. Conclusion and future work

The objective of conducting this study was to quantitatively analyze the fault density in design motifs at different levels: design, category, motif, and role. First, we measured and compared the fault density of participant classes and non-participant classes in design motifs. We found that there is no clear tendency for the difference in fault density between participant and non-participant classes. We then measured and compared the fault density of participant classes across the different categories of design motifs. We found that classes that participate in structural design motifs are less dense in faults than the other categories and less than non-participant classes as well. Next, we measured and compared the fault density of participant classes across design motifs. We found that only five design motifs show significant differences: Builder, Factory Method, Adapter, Composite, and Decorator. Classes that participate in each of these design motifs are less dense in faults than non-participant classes except for the Builder design motif. Classes that participate in the Builder are more dense in faults than non-participant classes. Finally, we measured and compared the fault density of participant classes across the different roles in each design motif. We found that only one pair of roles shows significant difference, which is Adapter vs. Client. In this pair, classes that participate in the Adapter role are less dense in faults than classes that participate in the Client role.

There are many directions for future works. First, we performed our analysis on five systems that are developed in Java. We cannot generalize our results to other systems that are developed in different programming languages. The different programming languages have different levels of expressiveness, which might affect the application of DPs. We must replicate our study on software systems that are developed in different programming languages.

Second, we performed our analysis on open-source software systems. These systems are developed by many developers from different backgrounds. Their levels of patterns experience are not controlled. This situation is different in commercial companies where the developers of software systems can be trained and prepared to work with DPs. So, we must replicate our study on commercial systems to improve the generalizability of the obtained results.

Third, our study was performed on the motifs of 17 GoF DPs out of 23 that exist in five systems. The other six patterns (i.e., Abstract Factory, Façade, Flyweight, Mediator, Chain-of-Responsibility, and Interpreter) are not applied in the subject systems that we used in this study. They should be addressed in the future. More motif instances and more subject systems can be considered as well.

Finally, we performed our study on motifs of object-oriented DPs. In the future, we think of addressing DPs that are developed in other paradigms, such as aspect-oriented paradigm [29] because the representations of these patterns are different from one paradigm to the other which might affect their applicability.

Acknowledgment

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum & Minerals (KFUPM) under Research Grant IN121056. We sincerely thank the anonymous reviewers for their insightful comments and suggestions, which have substantially improved the paper.

References

[1] T. Afacan, State Design Pattern Implementation of a DSP processor: a case study of TMS5416C, in: 6th IEEE International Symposium on Industrial Embedded Systems (SIES), 2011, 2011, pp. 67–70.
[2] M. Ali, M. Elish, A comparative literature survey of design patterns impact on software quality, in: 4th International Conference on Information Science and Applications (ICISA), 2013, pp. 1–7.
[3] A. Ampatzoglou, A. Kritikos, E.-M. Arvanitou, A. Gortzis, F. Chatziasimidis, I. Stamelos, An empirical investigation on the impact of design pattern application on computer game defects, in: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, Tampere, Finland, 2011.
[4] F. Arcelli Fontana, S. Maggioni, C. Raibulet, Understanding the relevance of micro-structures for design patterns detection, J. Syst. Softw. 84 (12) (2011) 2334–2347.
[5] F. Arcelli Fontana, M. Zanoni, A tool for design pattern detection and software architecture reconstruction, Inf. Sci. 181 (7) (2011) 1306–1324.
[6] L. Aversano, G. Canfora, L. Cerulo, C.D. Grosso, M.D. Penta, An empirical study on the evolution of design patterns, in: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Dubrovnik, Croatia, 2007.
[7] M.L. Bernardi, M. Cimitile, G.A. Di Lucca, A model-driven graph-matching approach for design pattern detection, in: 20th Working Conference on Reverse Engineering (WCRE), 2013, 2013 pp. 172–181.
[8] J. Bieman, G. Straw, H. Wang, P. Munger, R. Alexander, Design patterns and change proneness: an examination of five evolving systems, in: 9th International Software Metrics Symposium, 2003, pp. 40–49.

freepaper.me FREE paper

[9] C. Bunse, S. Stiemer, On the energy consumption of design patterns, in: 2nd Workshop Energy Aware Software-Engineering and Development, 2013.
[10] K. Chen, S.R. Schach, L. Yu, J. Offutt, G.Z. Heller, Open-source change logs, Empirical Softw. Eng. 9 (3) (2004) 197–210.
[11] A. De Lucia, V. Deufemia, C. Gravino, M. Risi, Behavioral pattern identification through visual language parsing and code instrumentation, in: 13th European Conference on Software Maintenance and Reengineering, 2009, CSMR '09, 2009, pp. 99–108.
[12] A. De Lucia, V. Deufemia, C. Gravino, M. Risi, Improving behavioral design pattern detection through model checking, in: 2010 14th European Conference on Software Maintenance and Reengineering (CSMR), 2010, pp. 176–185.
[13] J. Dietrich, C. Elgar, Towards a web of patterns, Web Semantics: Sci. Serv. Agents World Wide Web 5 (2) (2007) 108–116.
[14] K. El-Emam, S. Benlarbi, N. Goel, S. Rai, The confounding effect of class size on the validity of object-oriented metrics, IEEE Trans. Softw. Eng. 27 (2001) 630–650.
[15] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[16] J. Garzás, M. Piattini, Do rules and patterns affect design maintainability?, J Comput. Sci. Technol. 24 (2) (2009) 262–272.
[17] M. Gatrell, S. Counsell, Design patterns and fault-proneness a study of commercial C# software, in: 2011 Fifth International Conference on Research Challenges in Information Science (RCIS), 2011, pp. 1–8.
[18] M. Gatrell, S. Counsell, T. Hall, Design patterns and change proneness: a replication using proprietary C# software, in: 16th Working Conference on Reverse Engineering, 2009, WCRE '09, 2009, pp. 160–164.
[19] Y.-G. Guéhéneuc, P-mart: Pattern-like micro architecture repository, in: Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories, 2007.
[20] Y.-G. Guéhéneuc, Ptidej: Promoting patterns with patterns, in: Proceedings of the First ECOOP Workshop on Building a System using Patterns, Glasgow, UK, 2005.
[21] Y.-G. Guéhéneuc, J.-Y. Guyomarc'h, H. Sahraoui, Improving design-pattern identification: a new approach and an exploratory study, Softw. Qual. Control 18 (1) (2010) 145–174.
[22] Y.-G. Guéhéneuc, J. Guyomarch, K. Khosravi, H. Sahraoui, Design patterns as laws of quality, in: J. Garzas, M. Piattini (Eds.), Object-Oriented Design Knowledge: Principles, Heuristics and Best Practices, IGP, 2007, pp. 105–142.
[23] Y.-G. Guéhéneuc, H. Albin-Amiot, Using design patterns and constraints to automate the detection and correction of inter-class design defects, in: 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39, 2001, pp. 296–305.
[24] Y.-G. Guéhéneuc, G. Antoniol, DeMIMA: a multilayered approach for design pattern identification, IEEE Trans. Softw. Eng. 34 (5) (2008) 667–684.
[25] P. Hegedűs, D. Bán, R. Ferenc, T. Gyimóthy, Myth or reality? Analyzing the effect of design patterns on software maintainability, in: T.-H. Kim et al. (Eds.), Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity, vol. 340, Springer, Berlin, Heidelberg, 2012, pp. 138–145.
[26] D. Jing, D.S. Lad, Z. Yajing, DP-Miner: design pattern discovery using matrix, in: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2007, ECBS '07, 2007, pp. 371–380.
[27] N. Juristo, S. Vegas, Design patterns in software maintenance: an experiment replication at UPM – experiences with the RESER'11 joint replication project, in: 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011, 2011, pp. 7–14.
[28] F. Khomh, Y.-G. Guéhéneuc, G. Antoniol, Playing roles in design patterns: an empirical descriptive and analytic study, in: 25th IEEE International Conference on Software Maintenance (ICSM), 2009, pp. 83–92.
[29] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: M. Akşit, S. Matsuoka (Eds.), ECOOP'97 — Object-Oriented Programming, vol. 1241, Springer, Berlin Heidelberg, 1997, pp. 220–242.
[30] A. Koru, D. Zhang, K. ElEmam, H. Liu, An Investigation into the functional form of the size-defect relationship for software modules, IEEE Trans. Softw. Eng. 35 (2) (2009) 293–304.
[31] J.L. Krein, L.J. Pratt, A.B. Swenson, A.C. MacLean, C.D. Knutson, D.L. Eggett, Design patterns in software maintenance: an experiment replication at Brigham Young University", in: 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011, 2011, pp. 25–34.
[32] W.H. Kruskal, W.A. Wallis, Use of ranks in one-criterion variance analysis, J. Am. Stat. Assoc. 47 (1952) 583–621.
[33] A. Litke, K. Zotos, E. Chatzigeorgiou, G. Stephanides, Energy consumption analysis of design patterns, World Acad. Sci. Eng. Technol. (2005) 86–90.
[34] A. De Lucia, V. Deufemia, C. Gravino, M. Risi, Design pattern recovery through visual language parsing and source code analysis, J. Syst. Softw. 82 (7) (2009) 1177–1193.
[35] H.B. Mann, D.R. Whitney, On a Test of Whether One of Two Random Variables is Stochastically Larger Than the Other, Institute of Mathematical Statistics, 2000.
[36] A. Nanthaamornphong, J.C. Carver, Design patterns in software maintenance: an experiment replication at University of Alabama, in: 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011, 2011, pp. 15–24.
[37] J. Niere, W. Schafer, J. P. Wadsack, L. Wendehals, J. Welsh, Towards pattern-based design recovery, in: Proceedings of the 24rd International Conference on Software Engineering, 2002. ICSE 2002, 2002, pp. 338–348.
[38] S. Nija, R.A. Olsson, Reverse engineering of design patterns from Java source code, in: 21st IEEE/ACM International Conference on Automated Software Engineering, 2006, ASE '06, 2006, pp. 123–134.
[39] M.D. Penta, L. Cerulo, Y.-G. Guéhéneuc, G. Antoniol, An empirical study of the relationships between design pattern roles and class change proneness, in: 24th IEEE International Conference on Software Maintenance (ICSM), 2008, pp. 217–226.
[40] D. Posnett, C. Bird, P. Dévanbu, An empirical study on the influence of pattern roles on change-proneness, Empirical Softw. Eng. 16 (3) (2011) 396–423.
[41] L. Prechelt, M. Liesenberg, Design patterns in software maintenance: an experiment replication at Freie University; Berlin, in: 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011, 2011, pp. 1–6.
[42] L. Prechelt, B. Unger, W.F. Tichy, P. Brossler, L.G. Votta, A controlled experiment in maintenance: comparing design patterns to simpler solutions, IEEE Trans. Softw. Eng. 27 (12) (2001) 1134–1144.
[43] J. Rudzki, How design patterns affect application performance – a case of a multi-tier J2EE application, in: Proceedings of the 4th international conference on Scientific Engineering of Distributed Java Applications, Luxembourg-Kirchberg, Luxembourg, 2005.
[44] C. Sahin, F. Cayci, I. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, K. Winbladh, Initial explorations on design pattern energy usage, in: 1st International Workshop on Green and Sustainable Software, 2012, pp. 55–61.
[45] J.M. Smith, D. Stotts, SPQR: flexible automated design pattern extraction from source code, 2003, pp. 215–224.
[46] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S.T. Halkidis, Design pattern detection using similarity scoring, IEEE Trans. Softw. Eng. 32 (11) (2006) 896–909.
[47] M. Vokac, Defect frequency and design patterns: an empirical study of industrial code, IEEE Trans. Softw. Eng. 30 (12) (2004) 904–917.
[48] M. Vokac, W. Tichy, D. Sjoberg, E. Arisholm, M. Aldrin, A controlled experiment comparing the maintainability of programs designed with and without design patterns – a replication in a real programming environment, Empirical Softw. Eng. 9 (3) (2004) 149–195.
[49] C. Zhang, D. Budgen, A survey of experienced user perceptions about software design patterns, Inform. Softw. Technol. 55 (2013) 822–835.
[50] C. Zhang, D. Budgen, What do we know about the effectiveness of software design patterns?, IEEE Trans Softw. Eng. 38 (5) (2012) 1213–1231.

freepaper.me FREE paper