IET Software

Research Article

IET Journals
The Institution of Engineering and Technology

Methodology for the quantification of the effect of patterns and anti-patterns association on the software quality

ISSN 1751-8806
Received on 23rd March 2018
Revised 22nd March 2019
Accepted on 2nd April 2019
E-First on 5th July 2019
doi: 10.1049/iet-sen.2018.5087
www.ietdl.org

Shahid Hussain[1] ✉, Jacky Keung[2], Muhammad Khalid Sohail[1], Arif Ali Khan[3], Ghufran Ahmad[1], Muhammad Rafiq Mufti[4], Hasan Ali Khatak[1]

[1]*Department of CS, COMSATS University Islamabad, Pakistan*
[2]*Department of CS, City University of Hong Kong, Hong Kong*
[3]*Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu, 210016, People's Republic of China*
[4]*Department of CS, COMSATS University, Vehari Campus, Pakistan*
✉ *E-mail: shussain@comsats.edu.pk*

Abstract: The employment of design patterns is considered as a benchmark of software quality in terms of reducing the number of software faults. However, the quantification of the information about the hinder design issues such as the number of roles, type of design pattern, and their association with anti-pattern classes is still required. The authors propose a new methodology to evaluate the impact of certain design issues on the software quality in terms of quantification of fault density. Firstly, they mine the required information about the classes of each system under study. Secondly, they describe taxonomy to group the classes. Subsequently, they used statistical techniques to formulate and benchmark the results. They include the analysis of four open source projects with six design patterns and six anti-patterns in the case study. The main consequences are (i) the pattern participant classes are less dense in faults, (ii) the classes involved in the structural association between design patterns and anti-patterns are denser in faults, (iii) the pattern participant classes with multi-role and anti-pattern smell association is denser in faults as compared to others. The significant difference between fault density distributions of groups of classes is still unclear and required further empirical investigation.

1 Introduction

The design patterns are useful to provide a precise way for the investigation of quality attributes and maintenance of software [1]. In object-oriented software, the classes which are participated in design motif [2] are significantly different with minor precision from the non-participant classes in terms of frequency of faults [3, 4], several changes in order to fix the faults [3], and fault density [5]. The influential factors such as the type of design pattern, the involvement of a pattern participant class (PPC; participating in a design motif) in the number of roles [5, 6], the number of design pattern instances [7], and association of a design pattern with anti-patterns (or code) smell have a significant effect on the software quality [8, 9]. The relationship of the confounding effect of class size with the number of faults has been reported in many studies [10].

The fault density is coined [5] as a measure of a class, which is computed through the number of detected and confirmed faults in a class divided by the class size. Though, the fault density of pattern participant and non-participant classes (in design motif) has been quantified and differentiated at design motif, category, and the role levels. However, the quantification of the impact of influential factors is still required. For example, in XercesJ, the developer adds numerous interfaces signature in the class 'org.apache.xerces.validators.common.XMLValidator' without describing the actual purpose which might lead to SwissArmyknif anti-pattern [11]. The class 'org.apache.xerces.validators.common.XMLValidator' is also detected in the association (i.e. 'use-relationship') with 'org.apache.xerces.validators.dtd.DTDImporter' class which includes in command design motif in order to delegate the method calls. There is no clear tendency between participant and non-participant classes at design motif level [5], however, the participant classes of builders are denser in faults as compared to non-participant classes. Subsequently, PPCs with zero or one role are more stable as compared to those classes which are participating with more than one role. Through the influential factors such as the number of roles, type of design pattern, an association of design pattern and anti-patterns has been addressed in different studies and their impact on the software quality is investigated. However, the quantification of the fault density of the participant classes with the mutual effect of influential factors is still required to aid developers to rank the classes accordingly.

Rather than replicating the studies [3–5, 8], we propose a methodology to assess the fault density of object-oriented classes with respect to their influential factors. The procedure of the proposed methodology is described in steps as follows. Firstly, we employed two different tools named Percerons Client [http://www.percerons.com/] and defect detection for correction (DÉCOR) [12] to mine the classes participating in the design patterns (i.e. PPCs), non-participant classes, the number of roles in which PPCs are involved, and the detection of anti-patterns for each system under study. Secondly, we organised the classes into groups according to the proposed taxonomy shown in Fig. 1. We leverage the rules to describe the static relationship between design pattern P and anti-pattern AP [11]. [For a static relationship between design pattern P and anti-pattern AP, it is assumed that at least one class of P has a composition, aggregation, association, or use the relationship with one class of AP.] Thirdly, we follow the Runeson *et al.* [13] rules and performed a case study. The case study includes the analysis of four source projects named ArgoUML-0.26, XercesJ-2.4., JfreeChart-1.0.1, and PMD-1.8. Subsequently, in each analysis, we select six design patterns such as prototype and factory method from the creational, decorator and composite from the structural, and observer and command from the behavioural pattern category. These patterns have been analysed and reported as representative of complexity, data, and size problem [8, 11]. Subsequently, we used the set of six widely-known anti-patterns named as a number of AntiSingleton, Blob, LongMethod, LongParameterList, SwissArmyknif, and ComplexClass [14]. Finally, we used statistical techniques such as Mann–Whitney U and Kruskal–Wallis non-parametric tests and

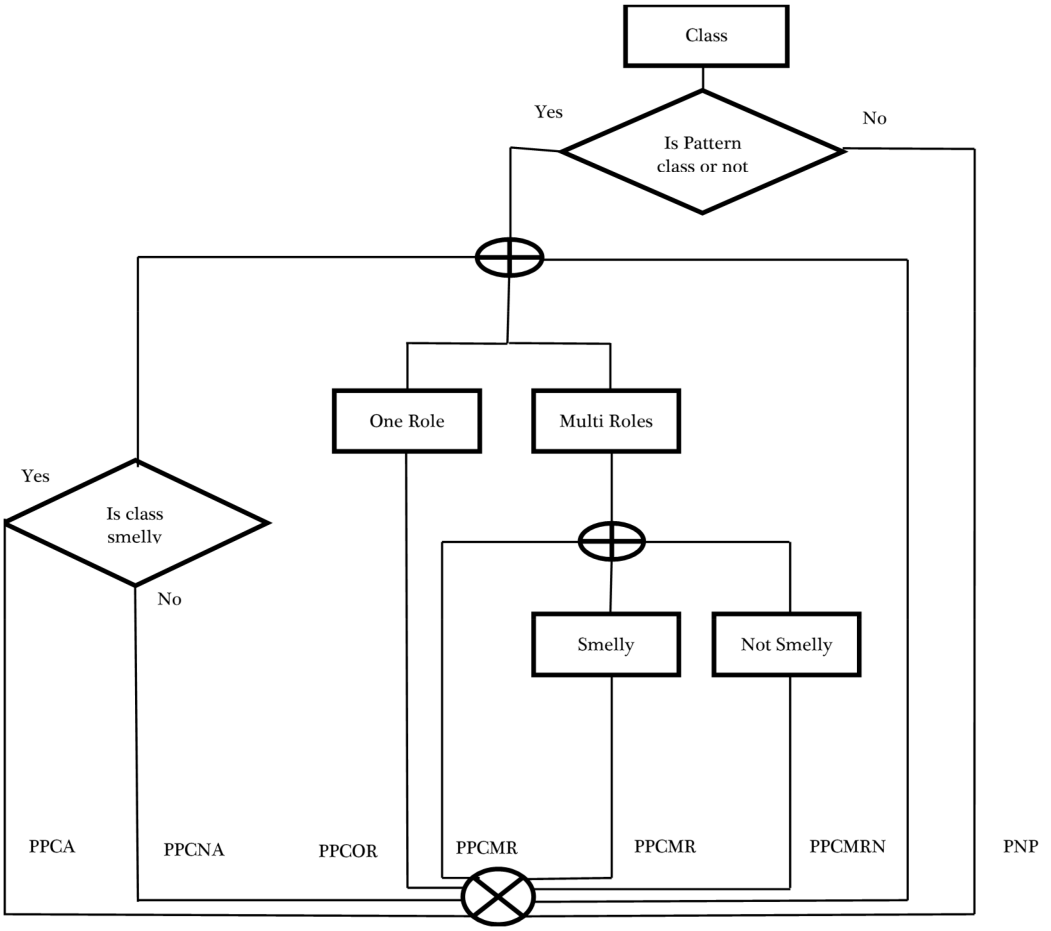

Fig. 1 *Taxonomy to group the classes for the assessment of fault density*

box-plot to assess the significant difference between groups in terms of fault density (as an assessor of software quality).

2 Related work

Since the introduction of the gang-of-four (GoF) design patterns, authors have claimed and recommended the use of design patterns to solve the design problems. Subsequently, researchers have made their efforts to empirically investigate the relationship between usage of design patterns and software quality using different proxies such as stability [6], fault density [5], several changes to fix the faults [15] and frequency of faults [3, 4]. Subsequently, the perception of developers/users [1], the interest of the research community [16], and the effectiveness of design patterns [17] can aid to describe a trade-off between employment of the GoF design patterns and the evolution of software. Vokac [3] presented a case study to describe the evolution and maintenance of a large commercial product named 'customer relationship management' written in C++ with ∼500,000 lines of code (LOC). In this case study, the authors target the defect frequency (i.e. defect rate) for the object-oriented classes, and comparatively investigate the PPCs with respect to the size of the code. Subsequently, in his study, Vokac consider the template method, factory method, decorator, singleton, and observer design patterns and reported (i) the high correlation of singleton and observer patterns with a large structure of a code, (ii) the compactness of the code using factory pattern causes lower defect rate, and (iii) the template method has no clear tendency either used in simple or complex situation.

Ampatzoglou *et al.* [4] present a case study, including the analysis of 97 Java-based open source game projects to investigate the relationship between design patterns and defect frequency. The open source projects which are included in the case study are varied with respect to the usage of design patterns, defect frequency, size and bug fixing efficiency. The authors performed an analysis on the singleton, adaptor, state, template method, prototype, abstract factory, composite, observer, strategy, proxy, and decorator design patterns. The main consequences of the author's work are (i) several design patterns are associated with the number of bugs, (ii) the adaptor design pattern is negatively associated with defect frequency, and the usage of observer design patterns may lead to decrease in the number of bugs. Consequently, the observer pattern is less error prone, and (iii) there is no clear tendency that design patterns are the root cause of the defect detection.

Moreover, in their subsequent study, Ampatzoglou *et al.* [6] present a case study, which includes 65,000 object-oriented classes related to Java-based open source projects and investigates the stability of classes which are participating in the instance of design patterns. In their study, the authors performed change impact analysis on the classes and analyse their four aspects, such as type of pattern, the role, the use of the corresponding pattern, and the application domain. The authors include an adapter, singleton, strategy-state, factory method, decorator, composite, visitor, template method, factory method, prototype, abstract factory, and observer design pattern in their presented case study. Subsequently, the authors reported that the classes associated with design patterns are shielded with any change and more stable. However, the classes with the coupled pattern occurrences are not more stable and required extra efforts during the testing process. Subsequently, the PPCs with a single role are more stable than PPCs with multi-role. The occurrence of classes with composite, singleton, fascade-mediator, decorator, and observer roles are more resistant to change propagation.

Mohammed [5] performed an empirical study to compare and calibrate the fault density at design, category, motif, and role levels. In their study, the authors used some statistical techniques to analyse and compare the fault density at certain levels. Moreover, the authors include abstract factory, builder, factory method, prototype, singleton, adapter, bridge, composite, decorator, fascade, flyweight, proxy, chain of responsibility, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor design patterns in each analysis. The main consequences of

their experimental work are (i) the factory method, composite, builder, decorator, and adapter are highly associated with fault density, (ii) the role pair adapter versus client present significant difference in terms of fault density, (iii) no clear difference between the fault density of participant and non-participant classes in a design motif, (iv) in terms of category, the builder design pattern is positively associated with fault density, (v) the decorator, factory method, composite, and adapter design patterns are negatively associated with fault density, and (vi) the classes with adapter role are less dense in faults as compared to client. Though, in these studies [3–6], the authors have investigated the effect of the number of roles and type of pattern participating classes on the software quality. However, the static association of PPCs with anti-pattern classes is not addressed.

The anti-patterns (due to the relationship between classes) and code smells (due to the inner working of classes) aid to describe the re-occurring software. In recent studies, Walter *et al.* [9], and Jaafar *et al.* [8, 11] have empirically investigated the static relationship between design patterns and anti-patterns (or code) smells and its effect during the evolution of software. Though the common objective of studies [8, 9, 11] is to explore the relationship between design patterns and anti-pattern (or code) smells. However, there are some hinder design issues such as the mutual influence anti-patterns (or code) smells, type of design pattern, and the number of roles associated with a PPC, which can affect the quantification of software quality attributes [6, 8, 9]. The aim of the proposed approach is to quantify the mutual effect of influential factors associated with PPCs.

3 Context of the proposed study

3.1 Systems under analysis

We used four open source systems namely ArgoUML-0.26, XercesJ-2.4, PMD-1.8, and JFreeChart-1.0.1. The brief description of each system is as follows.

3.1.1 ArgoUML-0.26: The ArgoUML-0.26 [http://ArgoUML-0.26.tigris.org/] is an open source modelling tool, which is introduced by J. E. Robbins at UC Irvine. The ArgoUML-0.26 is based on the unified modelling language (UML), operates on the Java platform, and described in ten languages. The support for the object constraint language and forward engineering are the two main features of ArgoUML-0.26.

3.1.2 XercesJ-2.4: The XercesJ-2.4 [https://xerces.apache.org/mirrors.cgi] is an apache collection of certain software libraries developed in Java language and used to parse, serialised, manipulate, and validate the XML documents.

3.1.3 PMD-1.8: The PMD-1.8 tool [https://sourceforge.net/p/pmd/news/2006/10/netbeans-plugin-v18-released/] is used to analyse the source code and find some programming flaws such as unused variables and unnecessarily created objects. Subsequently, PMD-1.8 also includes copy paste detector, which is used to identify the duplicate code in certain programming languages, such as C/C++, Java, C#, Ruby, PHP, MATLAB, Python and so on.

3.1.4 JfreeChart-1.0.1: The JfreeChart-1.0.1 [http://www.jfree.org/jfreechart/] is an open source chart library, which can aid developers in order to display the quality charts. The well-documented API output types (e.g. PNG and JPEG) and lesser general public license are the common features of JFreeChart-1.0.1 tool.

3.2 Design patterns under analysis

In this study, we consider six design patterns, which are taken from three categories namely creational, structural, and behavioural categories. The list of design patterns under study and their description is shown as follows:

- Factory method: The factory method design pattern is from the creational category. This pattern is used to define an interface for the creation of an object and allow subclasses that which class should be instantiated.
- Prototype: The prototype design pattern is from the creational category. This pattern is used to allow to create the customise objects and hide the information about their relevant classes.
- Decorator: The decorator design pattern is from the structural category. This pattern is used to allow the add responsibilities to an object dynamically and provide a flexible way for the use of subclasses in order to extend their functionality.
- Composite: The composite design pattern is from the structural category. This pattern is used to promote whole-part hierarchies and compose objects in a tree structure. Subsequently, it allows clients to treat individual objects and describe the uniform composition of objects.
- Observer: The observer design pattern is from the behavioural category. This pattern is used to define and visualise the one to many dependencies between objects. Consequently, the change in any one object is broadcast to all its dependent objects.
- Command: The command design pattern is from the behavioural category. This pattern is used to encapsulate the client's request (in the form of the method name and the value of the method's parameters) as an object. Moreover, it also promotes the invocation of a method on an object.

3.3 Anti-patterns under analysis

Though several anti-patterns have been catalogued [14], in this study, we used six commonly used anti-patterns namely long method, LongParameterList, AntiSingleton, Blob, ComplexClass and SwissArmyknif [8, 11]. The brief description of these anti-patterns is given as follows:

- Long method: The long method anti-pattern can be identified in a class via the existence of a long method in terms of LOC.
- LongParameterList: The LongParameterList anti-Pattern can be identified in a class through a method which has more than an average number of parameters (per method). The number of parameters is considered from the method definition across the system.
- AntiSingleton: The AntiSingleton anti-pattern can be identified in a class through the mutable class variables.
- Blob: The Blob anti-pattern can be identified through the existence of a large class, which is not cohesive. Blob class dominates most of the processing and decisions of the system.
- ComplexClass: The ComplexClass anti-pattern can be identified in a class via the existence of at least one long method in terms of LOC and cyclomatic complexity.
- SwissArmyknif: The SwissArmyknif anti-pattern is achieved when a designer wants to provide all possible uses of a class by designing a complex interface. It is also known as kitchen sink due to the use of thousands of method signatures in a single class.

4 Case study

In this study, we accomplish a case study that is an observational empirical method, which can be used to monitor the project relevant activities in a real-life context [13]. We present this case study, according to rules suggested by Runeson *et al.* [13]. In the domain of software engineering (SE), Runeson *et al.* present the design templates of two case studies namely holistic and embedded, which are further classified as a single case and multiple-case study. According to Runeson *et al.*, we noted that the holistic case study is that from which we can extract only one unit of analysis from each case, while in the case of embedded case study, multiple units of analysis can be extracted from a single case. In our study, we follow the holistic multiple-case study template and present the analysis process for achieving our research objective.

We performed analyses with four open source projects namely XercesJ-2.4, ArgoUML-0.26, PMD-1.8, and JFreeChart-1.0.1. SE

Table 1 Overview of the dataset structure

Variable	Type	Description
V1	demographic	name of the project
V2	demographic	name of the class
V3	continuous	LOC (i.e. size of the class)
V4	continuous	total faults
V5	continuous	fault density of class
V6	continuous	number of roles
V7	categorical	This is a dependent variable which presents the types of design patterns such as command, decorator, composite, factory method, observer, and prototype
V8	categorical	This is an independent variable which represents the group of the class. For example, PPC, PNPC, PPCOR, PPCMR, PPCAS, Pattern PPCNAS, PPCMRS, and PPCMRNS

research community has widely used these projects for the evaluation of design patterns and anti-patterns [4, 8, 11]. In each investigation, we considered a class as a case and unit of analysis. We leverage the idea of static relationship between design patterns P and anti-patterns AP [8, 11]. In this context, we analysed the static relationship between six GoF design patterns from the creational (factory method and prototype), structural (decorator and composite), and behavioural (command and observer) categories, and six anti-patterns (LongMethod, LongParameterList, AntiSingleton, Blob, ComplexClass and SwissArmyknif) [14].

We used Percerons Client tool for detection of pattern participant and non-participant classes besides the number of roles. Subsequently, in the case of anti-pattern detection, we used DÉCOR approach [12] via a widely known tool suit Ptidej along with DeMIMA [18]. The Ptidej tool suite [http://www.ptidej.net/tools/] used the built PADL meta model by parsing the object-oriented source code. The source code is parsed into their components such as class, member class, interface, member interface, attributes, methods, and inheritance. The Ptidej tool is coded in java language and its architecture is described via UML class diagram. The Ptidej architecture is described in three layers. The first layer namely Common Ptidej Library is used to describe the utility and libraries classes such as CFParese. The second layer namely PADL is used to describe the meta models for programme and design motif. For example, padl. visitor package is providing the list of defaults visitor's pattern classes. The third layer has several independent projects such as parsers of java classes and primitives, operators, metrics as a computational framework.

Subsequently, we used the concurrent versions system (CSV) files rather than the release notes of each subject system hosted on SouceForge. We look at the error, defect, flaw, bug, and fault keywords to investigate the commits of each class (excluding the nested classes such as inner classes) and count the number of faults for the corresponding class. Finally, we evaluate the fault density of each class of the target system. The dataset structure shown in Table 1 is used to record the information of each class. The V8 attribute of the dataset (shown in Table 1) used to group the classes into eight categories on the basis of a number of roles and anti-pattern association of PPCs. Subsequently, the classification of classes into eight categories is shown in Fig. 1. The groups PPC and pattern non-participant class (PNPC) are used to classify the classes either as pattern participant or not. The groups PPC with one role (PPCOR) and PPC with multi-roles (PPCMRs) are used to classify the PPCs with respect to a number of roles. The groups PPC with anti-pattern smell (PPCAS) and PPC with no anti-pattern smell (PPCNAS) are used to classify the PPCs either they are associated with at least one anti-pattern smell or not. Finally, the groups PPCMRS and PPC with multi-roles and not smelly (PPCMRNS) classify the multi-role PPCs either they are associated with at least one anti-pattern smell or not. Subsequently, groups are statistically analysed in terms of V5 (i.e. fault density) of Table 1.

4.1 Research objectives and research questions (RQs)

The primary objective of our study is to empirically investigate the groups (classified via the proposed taxonomy in Fig. 1 of classes with respect to (i) their association with design patterns, (ii) their association with anti-patterns, and (iii) the number of roles. Subsequently, the fault density measure (as a proxy of software quality) is used to benchmark the quality level between groups.

We assume P and AP are the set of design patterns and anti-pattern types, respectively. Subsequently, C present the set of classes for a target system. The terms association, composition, use, and aggregation are used to describe the static relationship between classes. Equations (1)–(3) are used to describe the association of a class c (i.e. $c \in C$) with a design pattern p (i.e. $p \in P$), or an anti-pattern ap (i.e. $ap \in AP$) or both [11, 19]

$$c = f(p), \quad \forall\, c \in C,\ p \in P, \tag{1}$$

$$c = f(ap), \quad \forall\, c \in C,\ ap \in AP, \tag{2}$$

$$c = f(p,\ ap), \quad \forall\, c \in C,\ p \in P,\ ap \in AP\,. \tag{3}$$

Through our objective statement, we extract the following research questions:

RQ-1. Is the quality level of pattern participant and non-participant classes the same in terms of fault density?
The aim of RQ-1 is to investigate that either PPCs are denser in faults as compared to non-participant classes or not. Consequently, quality of PPC and PNPC groups is analysed in terms of fault density.
RQ-2. Is the quality level of PPCs involved in one or more than one role same in terms of fault density?The aim of RQ-2 is to investigate that either PPCs involved in more than one role are denser in faults as compared to those PPCs which are performing a single role. Consequently, the quality of PPCOR and PPCMR groups is analysed in terms of fault density.
RQ-3. Is the quality level of PPCs, which are either or not involved in static association with at least one anti-pattern same in terms of fault density?The aim of RQ-3 is to investigate that either pattern participants classes associated with at least one anti-pattern are denser in faults as compared to those pattern participants' classes which have no such association. Consequently, quality of PPCAS and PPCNAS groups are analysed in terms of fault density.
RQ-4. Is the quality level of multi-role PPCs, which are either or not involved in static association with at least one anti-pattern same in terms of fault density?The aim of RQ-4 is to investigate that either multi-role pattern participants classes associated with at least one anti-pattern are denser in faults as compared to those multi-role pattern participants' classes, which have no such association. Consequently, the quality of PPCMRS and PPCMRNS groups is analysed in terms of fault density.
RQ-5. Is the quality level of PPCs which have static associations with at least one anti-pattern class same for different types of design patterns in terms of fault density?

The aim of RQ-5 is to investigate that which one type of pattern participants classes associated with at least one anti-pattern is denser in faults. Subsequently, we group the pattern participants classes with respect to V7 attribute of Table 1.

4.2 Data collection and used tools

We employed two different tools named Percerons Client and DÉCOR (incorporated into the Ptidej tool suite) to retrieve and record the information of classes including the number of roles, type of design pattern and anti-pattern. Subsequently, we used the CSV files of the systems under study and record the data (Table 1).

Table 2 Shapiro–Wilk and Kolmogorov–Smirnov test results for normality of groups distribution

Projects	Shapiro–Wilk				Kolmogorov–Smirnov			
	PPC versus PNPC	PPCOR versus PPCMR	PPCAS versus PPCNAS	PPCMRS versus PPCMRNS	PPC versus PNPC	PPCOR versus PPCMR	PPCAS versus PPCNAS	PPCMRS versus PPCMRNS
ArgoUML-0.26	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
JFreeChart-1.0.1	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
PMD-1.8	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
XercesJ-2.4	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05

4.3 Hypothesis formulation

The core component of SE experiment is to formalise the hypothesis. The list of formulating hypotheses of our study is as follows.

HP-1. The fault density distributions of PPC is significantly different from the fault density distribution of PNPC.
HP-2. The fault density distributions of PPCOR are significantly different from the fault density distribution of PPCMR.
HP-3. The fault density distributions of PPCAS are significantly different from the fault density distribution of PPCNAS.
HP-4. The fault density distributions of PPCMRSs are significantly different from the fault density distribution of PPCMRNS
HP-5. The fault density distribution of PPC is significantly different with respect to type of pattern *P*.

4.4 Data analysis

In order to achieve our research objective and respond to the research questions, we used the following steps.

Step-1. We used tools (Section 4.2) to record data.
Step-2. We performed Mann–Whitney *U* (non-parametric) [14] test in order to compare the fault density distributions difference of two groups and rank the quality levels, For example, PPC and PNPC. The assumptions to perform Mann–Whitney *U* test are (a) V8 (Table 1) should be categorical, (b) V5 (Table 1) might be ordinal or continuous, (c) independent observations between groups, e.g. between pattern and non-pattern (PPC and PNPC for the V8 in Table 1) classes, and (d) both groups (i.e. PPC and PNPC) should not be normally distributed. Subsequently, we performed Kolmogorov–Smirnov and Shapiro–Wilk tests to evaluate the normality of data of both groups by considering an assumption 'the data is not normally distributed' with *p*-value <0.05 (respond to RQ-1, RQ-2, RQ-3, and RQ-4).
Step-3. We performed the Kruskal–Wallis (non-parametric) [20] test to compare the fault density distribution difference among more than two groups. For example, the fault density distributions between command, decorator, composite, factory method, observer, and prototype design pattern instances. The assumptions of the Kruskal–Wallis test are (a) V7 (Table 1) should be categorical and V5 (Table 1) might be ordinal or continuous (respond to RQ-5).
Step-4. We drew box-plots to visualise and calibrate the fault density distributions between the groups. In order to significantly differentiate the distributions of the sample groups (e.g. PPC versus PNPC group of classes), we used the distance between medians (DBM) and overall visible spread (OVS) measures.

We calibrate the results according to assumptions which are considered to compare the data distribution between two samples via a box plot. The Q_1, Q_2, and Q_3 are the common notations of box-plots used to compute the lower, median, and upper quartile of the sample data. Subsequently, these notations aid to describe the size (i.e. inter quartile range (IQR)), whisker, and position of the median in box-plot. The main assumptions which we considered during the analysis to respond to our research questions are as follows:

Assumption 1: The fault density distribution between two groups of classes is considered different when their box-plots do not overlap each other. Consequently, the difference between two sample distributions can be recommended. Suppose PPC and PNPC are the two sample distributions via fault density attribute V5 (Table 1). The assumption is denoted through (4).

$$Q_1(\text{PPC}) > Q_3(\text{PNPC}) \vee Q_1(\text{PNPC}) > Q_3(\text{PPC}). \tag{4}$$

Assumption 2: The fault density distribution between two groups of classes will be the same when their box-plots overlap each other via medians, shown in (5)

$$\begin{aligned} &Q_2(\text{PPC}) > Q_2(\text{PNPC}) \wedge Q_2(\text{PPC}) < Q_3(\text{PPC}) \\ &\quad \vee Q_2(\text{PPC})\langle Q_2(\text{PNPC}) \wedge Q_2(\text{PPC})\rangle Q_1(\text{PPC}). \end{aligned} \tag{5}$$

Assumption 3: The fault density distribution between two groups of classes will be the same when their box-plots overlap each other without medians. In this case, the sub-assumptions are as follows.

Assumption-3.1: The variability of fault density distribution between two groups of classes (i.e. PPC and PNPC) will be the same when their IQR is the same (6)

$$\text{IQR}(\text{PPC}) = \text{IQR}(\text{PNPC}). \tag{6}$$

However, it could not define that underlying fault density distribution is symmetric.
Assumption-3.2: The underlying fault density distribution will be symmetric if the difference of Q_1 and Q_3 with Q_2 (median) is the same, shown in (7). The same assumption can be done between whiskers and Q_2

$$\text{diff}(Q_1, Q_2) = \text{diff}(Q_3, Q_2). \tag{7}$$

4.5 Results and discussion

In order to respond to RQ-1, RQ-2, RQ-3, and RQ-4, firstly we classified the information about classes (Table 1) into different groups according to their labels (V8 in Table 1) for each system under study. For example, groups PPC (and PNPCs are created for ArgoUML-0.26, FreeChart, PMD-1.8, and XercesJ-2.4, respectively. Secondly, before applying the Mann–Whitney *U* test, we performed Kolmogorov–Smirnov and Shapiro–Wilk (non-parametric) tests to evaluate the normality of data between the two groups. The results of Kolmogorov–Smirnov and Shapiro–Wilk test for the groups of each system are shown in Table 2. The result of Table 2 indicates that group data for each system is not normally distributed (with *p*-value <0.05 used to reject null hypothesis i.e. H_0: data of each group is significantly different from normality). For example, Shapiro–Wilk and Kolmogorov–Smirnov test values (i.e. *p*-values) for PPC and PNPC (i.e. *p*-value <0.05) indicate that data distribution for PPC and PNPC are significantly differenced from its normality. Third, we performed the Mann–Whitney *U* test to compare the difference between the fault density distributions of the groups for each system under study.

From the results of Table 3, we do not claim the significant difference between groups across the systems under study. Though we observed the significant difference between the fault density

Table 3 Results of Mann–Whitney *U* test for systems under study

Analysis	PPC versus PNPC	PPCOR versus PPCMR	PPCAS versus PPCNAS	PPCMRS versus PPCMRNS
ArgoUML-0.26	0.2213	0.1342	0.4521	0.0522
JFreeChart-1.0.1	0.0335	0.3723	0.0231	0.1342
PMD-1.8	0.0493	0.0354	0.3231	0.0432
XercesJ-2.4	0.0323	0.0371	0.0223	0.0232

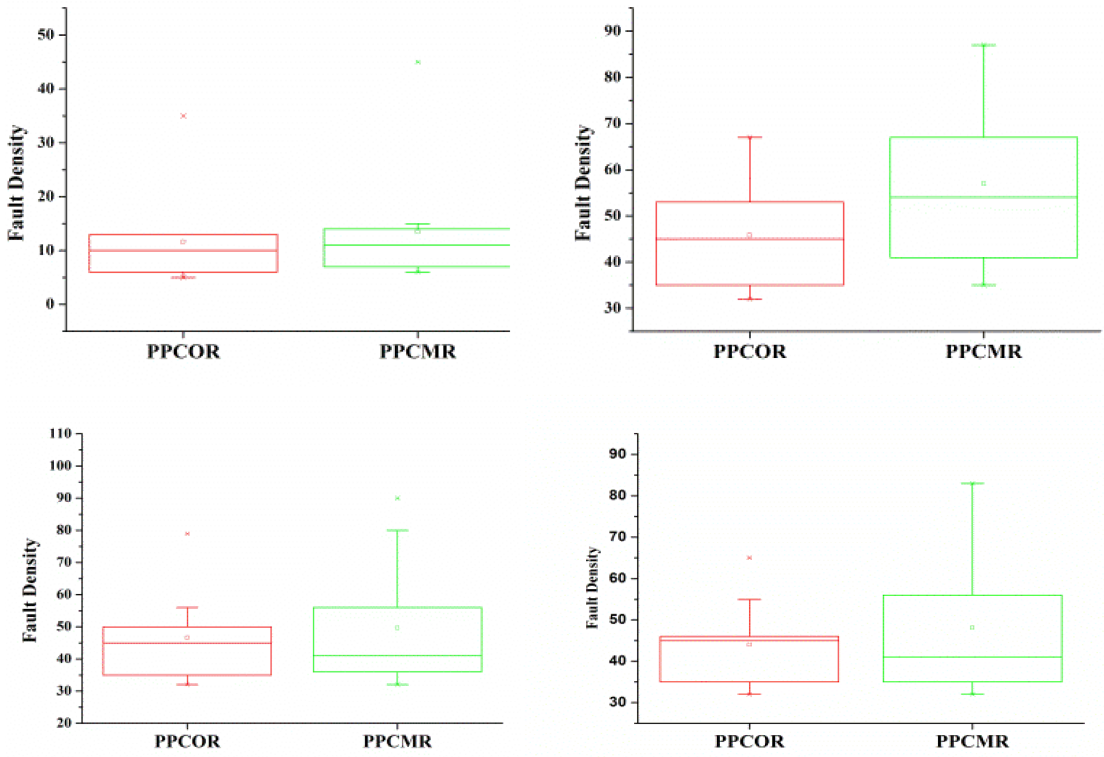

Fig. 2 *Fault density distributions between groups PPCOR and PPCMR for* ***(a)*** ArgoUML-0.26, ***(b)*** JFreeChat-1.0.1, ***(c)*** PMD-1.8 and ***(d)*** XercesJ-2.4, respectively

distributions of groups PPC and PNPC (p-value = 0.0323), PPCOR and PPCMR (p-value = 0.0371), PPCAS and PPCNAS (p-value = 0.0223), and PPCMRS and PPCMRNS (p-value = 0.0232) for XercesJ-2.4. However, for ArgoUML-0.26, PMD-1.8, and JFreeChart-1.0.1, we do not observe the significant difference between all groups. For example, in case of PMD-1.8, we observed a significant difference between groups except for PPCAS and PPCNAS with p-value >0.05, which describe the rejection of null hypothesis (i.e. H_0: data distribution of groups is a significant difference).

Subsequently, in order to respond to the research questions, we drew the box-plots to visualise and calibrate the fault density distribution between groups, such as PPC and PNPC (i.e. respond to RQ-1), PPCOR and PPCMR (i.e. respond to RQ-2), PPCAS and PPCNAS (i.e. respond to RQ-3), and PPCMRS and PPCMRNS (i.e. respond to RQ-4). The IQR (size of the box), the position of medians (i.e. Q_2), and size of whisker aid us to calibrate the fault density between groups by considering assumptions (about interpretation through box-plot) discussed in Section 4.4.

For example, in case of PPCOR and PPCMR groups of JFreeChart-1.0.1 (Fig. 2*b*), the IQR of PPCMR (i.e. 25) is greater than the IQR of PPCOR (i.e. 10), which indicate that the group of pattern participant classes with single role is less dense in faults as compared to the group of pattern participant classes involved in multi-roles. Subsequently, in case of PPCAS and PPCNAS for PMD-1.8 (Fig. 3*c*), though the IQR for both groups PPCAS (i.e. IQR = 15) and PPCNAS (i.e. IQR = 14.5) is the same as a minor difference. However, the size of whisker aids us to determine that the group of pattern participant classes, which have a static association with anti-smells, is denser in faults as compared to another group of classes. Moreover, in case of PPC and PNPC group of ArgoUML-0.26 (Fig. 4*a*), though the IQR and size of whisker are approximately the same, however, the position of medians for PPC (i.e. 10) and PNPC (i.e. 11) aid us to indicate that groups of pattern participant classes are less dense in faults as compared to non-participant classes.

Finally, through the IQR, the position of medians (i.e. Q2), and size of whisker aid us to interpret the difference between two groups of classes. However, in order to benchmark the difference between groups, we applied two widely used measures DBM and OVS [21]. The aim of DBM and OVS is to calibrate and benchmark the fault density distributions between groups of the classes. According to their fractional percentage (i.e. DBM/OVS × 100), the significant difference between groups can be decided with respect to sample size. For example, when the sample size is 30 then the fractional percentages >33% indicate the significant difference between groups. Subsequently, the fractional percentage 20 and 10% recommended when the sample size is ≥100 and 1000 observations, respectively. The sample size of systems under study is between 100 and 1000. Consequently, the results are benchmarked according to the fractional percentage of 20%. The analysis results are shown in Table 4.

Respond to RQ-1. The aim of RQ-1 is to investigate the differences between two groups of classes namely PPC and PNPCs. The main consequences of the results of Table 3 and Figs. 4*a–d* are as follows:

- We could not observe the significant difference between PPC and PNPC groups of XercesJ-2.4 (Diff (%) = 0) in terms of fractional percentage (i.e. DBM/OVS × 100) of DBM and OVS measures.
- We observed a significant difference between PPC and PNPC groups of PMD-1.8 (Diff (%) = 26.67), ArgoUML-0.26 (Diff

(%) = 23.08) and JFreeChart-1.0.1 (Diff (%) = 22.22) in terms of fractional percentage of DBM and OVS measures.

The OVS between fault density distributions of groups PPC and PNPC remain high for PMD-1.8 as compared to other systems.

The results of the Mann–Whitney *U* test (Table 3) and implication derived from the box plots (Figs. 4*a–d*) for each system

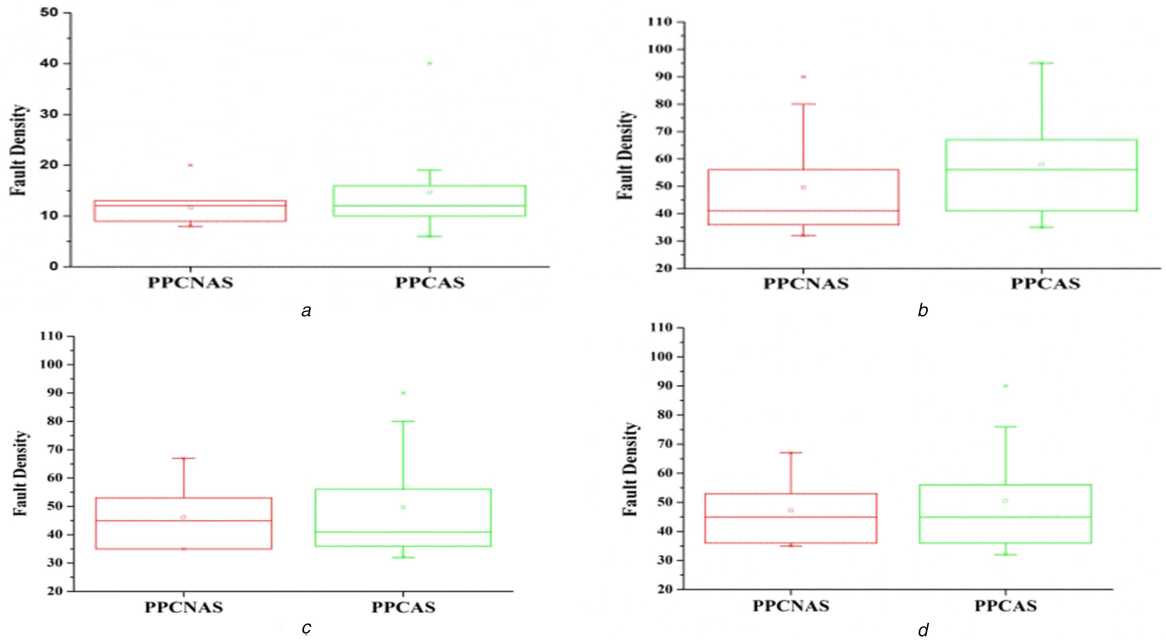

Fig. 3 *Fault density distributions between groups PPCNAS and PPCAS for* ***(a)*** ArgoUML-0.26, ***(b)*** JFreeChat-1.0.1, ***(c)*** PMD-1.8 and ***(d)*** XercesJ-2.4, respectively

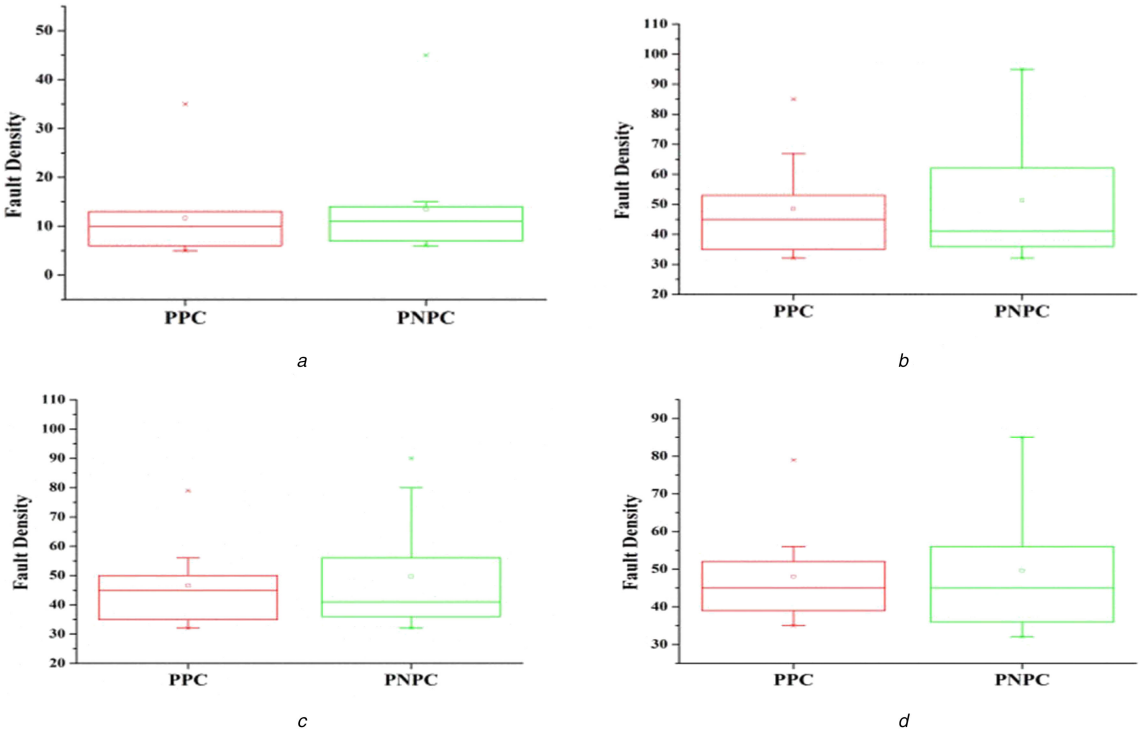

Fig. 4 *Fault density distributions between groups PPC and PNPC for* ***(a)*** ArgoUML-0.26, ***(b)*** JFreeChat-1.0.1, ***(c)*** PMD-1.8 and ***(d)*** XercesJ-2.4, respectively

Table 4 Assessment and benchmarking of differences between fault density distributions of groups for each system using DBM and OVS and their fractional percentage

Groups	ArgoUML-0.26			JFreeChart-1.0.1			PMD-1.8			XercesJ-2.4		
	DBM	OVS	Diff(%)	DBM	OVS	Diff., %	DBM	OVS	Diff., %	DBM	OVS	Diff., %
PPC versus PNPC	1.5	6.5	23.08	4	18	22.22	4	15	26.67	0	13	0
PPCOR versus PPCMR	0.5	6.5	7.69	17	30.5	55.74	3	17.5	17.14	7.5	17.5	42.86
PPCAS versus PPCNAS	0.5	5	10	16	25.5	62.75	3	28	10.71	4.5	14	32.14
PPCMRS versus PPCMRNS	2.5	5.5	45.4	3	7.5	17.14	13	25	52	34.5	40	86.25

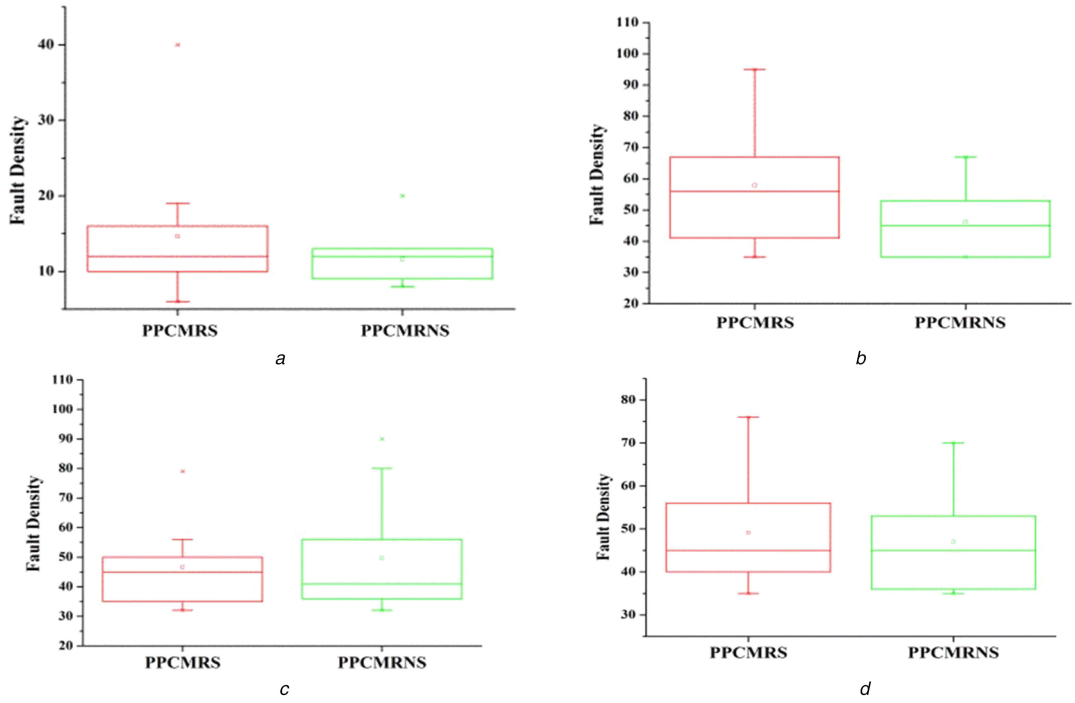

Fig. 5 *Fault density distributions between groups PPCMRS and PPCMRNS for* ***(a)*** ArgoUML-0.26, ***(b)*** JFreeChat-1.0.1, ***(c)*** PMD-1.8 and ***(d)*** XercesJ-2.4, respectively

indicate the validity of our hypothesis (HP-1). On the basis of the validity of HP-1, the answer to RQ-2 is that the fault density distributions between groups PPC and PNPC are significantly different. However, it depends on the sample size and the system under study.

Respond to RQ-2: The aim of RQ-2 is to investigate the differences between two groups of classes namely PPCOR and PPCMR. The main consequences of the results of Table 3 and Figs. 2*a–d* are as follows:

- We could not observe the significant difference between PPCOR and PPCMR groups of ArgoUML-0.26 (Diff(%) = 7.69) and PMD-1.8 (Diff (%) = 17.14) in terms of fractional percentage (i.e. DBM/OVS × 100) of DBM and OVS measures.
- We observed a significant difference between PPCOR and PPCMR groups of JfreeChart-1.0.1 (Diff (%) = 55.74) and XercessJ-2.4 (Diff(%) = 42.46) in terms of fractional percentage of DBM and OVS measures.
- The OVS of fault density distribution between groups PPCOR and PPCMR remains high for JfreeChart-1.0.1 as compared to other systems.

The results of the Mann–Whitney *U* test (Table 3) and implication derived from box plots (Figs. 2*a–d*) for each system indicate the validity of our hypothesis (HP-2). On the basis of the validity of HP-2, the answer to RQ-2 is that the distribution of fault density distributions between PPCOR and PPCMR is significantly different. However, it depends on the sample size and the system under study.

Respond to RQ-3: The aim of RQ-3 is to investigate the differences between two groups of classes namely PPCAS and PPCNAS. The main consequences of the results of Table 3 and Figs. 3*a–d* are as follows:

- We could not observe the significant difference between the PPCAS and PPCNAS groups of ArgoUML-0.26 (Diff (%) = 10) and PMD-1.8 (Diff (%) = 10.71) in terms of fractional percentage (i.e. DBM/OVS × 100) of DBM and OVS measures.
- We observed a significant difference between PPCAS and PPCNAS groups of JfreeChart-1.0.1 (Diff (%) = 62.75) and XercessJ-2.4 (Diff (%) = 32.14) in terms of fractional percentage of DBM and OVS measures.

The OVS of fault density distribution between groups PPCAS and PPCNAS remains high for JfreeChart-1.0.1 as compared to other systems.

The results of the Mann–Whitney *U* test (Table 3) and implication derived from the box plots (Figs. 3*a–d*) for each system indicates the validity of our hypothesis (HP-3). On the basis of the validity of HP-3, the answer to RQ-3 is that the distribution of fault density distributions between PPCAS and PPCNAS is significantly different, however, it depends on the sample size and the system under study.

Respond to RQ-4: The aim of RQ-4 is to investigate the differences between two groups of classes namely PPCMRS and PPCMRNS. The main consequences of the results of Table 3 and Figs. 5*a–d* are as follows:

- We could not observe the significant difference between PPCMRS and PPCMRNS groups of JFreeChart-1.0.1 (Diff (%) = 17.14) in terms of fractional percentage (i.e. DBM/OVS × 100) of DBM and OVS measures.
- We observed a significant difference between PPCMRS and PPCMRNS groups of ArgoUML-0.26 (Diff (%) = 45.40), PMD-1.8 (Diff (%) = 52), and XercesJ-4.2 (Diff (%) = 86.25) in terms of fractional percentage of DBM and OVS measures.
- The OVS of fault density distribution between groups PPCMRS and PPCMRNS remains high for XercesJ-4.2 (Diff (%) = 86.25) as compared to other systems.

The results of the Mann–Whitney *U* test (Table 3) and implication derived from the box plots (Figs. 5*a–d*) for each system indicates the validity of our hypothesis (HP-4). On the basis of the validity of HP-3, the answer to RQ-4 is that the distribution of fault density distributions between PPCMRS and PPCMRNS is significantly different, however, it depends on the sample size and the system under study.

Respond to RQ-5: The aim of RQ-5 is to investigate the fault density distribution between the classes participated in design patterns and have a static association with at least one anti-pattern class. In this regard, we merge the pattern participated classes of all systems and perform analysis accordingly. Firstly, we group the

pattern participated classes according to their pattern type (i.e. composite, factory method, command, decorator, observer, and prototype), which have an association with at least one anti-pattern class. Secondly, we apply the Bonferroni correction test to assess the normality (i.e. p-value <0.05) of data (i.e. fault density) of each pattern group.

Third, we apply the Kruskal–Wallis test to compare the difference between the fault density distribution of groups and obtained the significant with p-value = 0.032 (i.e. <0.05). Though the significant difference (with p-value = 0.032) between fault density distributions of groups (with respect to type design pattern) is observed. However, the significance of the association between design pattern and anti-pattern varies with respect to the type of pattern and anti-patterns. These results indicate the validity of our hypothesis (HP-5). On the basis of the validity of HP-5, the answer to RQ-5 is that there is a significant difference between fault density distributions of groups with respect to the type of design pattern.

5 Implications of the proposed study

The calibration of software quality is the challenging task for a developer which can aid in ranking the classes, and reduce the effort and time required to perform maintenance activities. The aim of the proposed study is to introduce taxonomy to group the classes with respect to their influential factors such as the number of roles, type of design pattern, and static associations between the design pattern and anti-pattern. The main consequences of the proposed empirical study are as follows:

- The classes which have an association with a design pattern are less dense in faults as compared to those classes which have a lack of association.
- There is no clear tendency that pattern participant classes, which perform a single role, are less dense in faults as compared to multi-roles pattern participant classes. For example, for ArgoUML-0.26 and PMD-1.8, the fault density distributions of PPCOR and PPCMR groups are approximately the same.
- Though, for JFreeChart-1.0.1 and Xercess-2.4, we observe the clear tendency that the pattern participant classes associated with anti-pattern are denser in faults as compared to others. However, this assumption is not true for ArgoUML-0.26 and PMD-1.8. The reason behind the observation might be the type of design pattern and anti-pattern, which is detected during the empirical investigation.
- We observed that multi-role pattern participant classes associated with at least one anti-pattern class are denser in fault as compared to others.
- The fault density distribution between groups of classes with respect to the type of design pattern is significantly different for the systems under study. However, experimental results might be altered by considering other types of design patterns and anti-patterns, and system in the proposed study.

6 Conclusion

The objective of the proposed study is to empirically investigate the fault density (referred as a proxy of the software quality) distributions between groups of classes with their influential factors such as type of design pattern, the involvement of the pattern participant classes with different number of roles, and static association with anti-pattern classes. We conduct this case study on four systems named ArgoUML-0.26, JFreeChat-1.0.1, PMD-1.8, and XercesJ-2.4, six design patterns, and six anti-patterns. In each analysis, we grouped the classes according to the proposed taxonomy, compute the fault density and calibrate it in groups by applying statistical techniques and tests. We observed some consequences. First, there are no clear significant differences in fault density distributions between groups organised through the proposed taxonomy. Second, the static association with design pattern and anti-pattern classes affects the software quality. Third, the association of pattern participant classes with anti-pattern classes, and their involvement in multi-roles can also affect the software quality. Finally, we found the significant difference between the fault density distributions between the groups of classes (associated with at least one anti-pattern class) with respect to type design pattern. However, observations might vary due to the type of design pattern and anti-pattern detected during the investigation. In the future, we will consider other types of design patterns and anti-patterns and more systems to investigate the effectiveness of the proposed approach.

7 References

[1] Zhang, C., Budgen, D.: 'A survey of experienced user perceptions about software design patterns', *Inf. Softw. Technol.*, 2013, **55**, (5), pp. 822–835

[2] Gamma, E., Richard, H., Johnson, R., *et al.*: '*Design patterns: elements of reusable object-oriented software*' (Addison-Wesley, Boston, 1995)

[3] Vokac, M.: 'Defect frequency and design patterns: an empirical study of industrial code', *IEEE Trans. Softw. Eng.*, 2004, **30**, (12), pp. 904–917

[4] Ampatzoglou, A., Kritikos, A., Arvanitou, E-M., *et al.*: 'An empirical investigation on the impact of design pattern application on computer game defects'. Proc. 15th Int. Academic MindTrek Conf., Finland, 2011

[5] Elish, M.O., Mohammed, M.A.: 'Quantitative analysis of fault density in design patterns', *Inf. Softw. Technol.*, 2015, **66**, pp. 58–72

[6] Ampatzoglou, A., Chatzigeorgiou, A., Charalampidou, S., *et al.*: 'The effect of GoF design patterns on stability: a case study', *IEEE Trans. Softw. Eng.*, 2015, **41**, (8), pp. 781–802

[7] Sfetsos, P., Ampatzoglou, A., Chatzigeorgiou, A., *et al.*: 'A comparative study on the effectiveness of patterns in software libraries and standalone applications'. 9th Int. Conf. on the Quality of Information and Communication Technology, Guimaraes, Portugal, 2014

[8] Jaafar, F., Gueheneuc, Y-G., Hamel, S., *et al.*: 'Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults', *Empir. Softw. Eng.*, 2016, **21**, pp. 896–931

[9] Walter, B., Alkhaeir, T.: 'The relationship between design patterns and code smells: an exploratory study', *J. Inf. Softw. Technol.*, 2016, **74**, pp. 127–142

[10] Zhou, Y., Xu, B., Leung, H, *et al.*: 'An in-depth study of the potentially confounding effect of class size in fault prediction', *ACM Trans. Softw. Eng. Methodol.*, 2014, **23**, (1), pp. 1–51

[11] Jaafar F. Gueheneuc, Y-G., Hamel S, , *et al.*: 'Analysing anti-patterns static relationships with design patterns', *Electron. Commun. EASST*, 2013, **59**, pp. 1–26

[12] Moh, N., Gueheneuc, Y.G., Duchien, L., *et al.*: 'DECOR: A method for the specification and detection of code and design smells', *IEEE Trans. Softw. Eng.*, 2010, **36**, (1), pp. 20–36

[13] Runeson, P., Host, M., Rainder, A., *et al.*: '*Case study research in software engineering: guidelines and examples*' (Wiley & Sons, Hoboken, NJ, USA, 2012)

[14] Brown, W.H., Malveau, R.C., McCormick, H.W., *et al.*: '*Anti patterns: refactoring software, architectures, and projects in crisis*' (Wiley, New York, NY, USA, 1998, 1st edn.)

[15] Gatrell, M., Counsell, S.: 'Design patterns and fault-proneness a study of commercial C# software'. Fifth Int. Conf. on Research Challenges in Information Science (RCIS), Gosier, Guadeloupe, 2011, pp. 1–8

[16] Ampatzoglou, A., Charalampidou, S., Stamelos, I.: 'Research state of the art on GoF design patterns: a mapping study', *J. Syst. Softw.*, 2013, **86**, pp. 1945–1964

[17] Zhang, C., Budgen, D.: 'What do we know about the effectiveness of software design patterns?', *IEEE Trans. Softw. Eng.*, 2012, **38**, (5), pp. 1213–1231

[18] Gueheneuc, Y.G., Antoniol, G.: 'DeMIMA: a multi-layered framework for design pattern identification', *IEEE Trans. Softw. Eng.*, 2008, **34**, (5), pp. 667–684

[19] Gueheneuc, Y-G., Amiot, H.A.: 'Recovering binary class relationships: putting icing on the UML cake'. Proc. 19th Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Vancouver, Canada, 2004, pp. 301–314

[20] Kruskal, W.H., Wallis, W.A.: 'Use of ranks in one-criterion variance analysis', *J. Am. Stat. Assoc.*, 1952, **47**, pp. 583–621

[21] Wild, C.J., Pfannkuch, M., Regan, M., *et al.*: 'Towards more accessible conceptions of statistical inference', *J. R. Stat. Soc. A*, 2011, **174**, (2), pp. 247–295