The Effect of Gang-of-Four Design Patterns Usage on Design Quality Attributes

Shahid Hussain
Department of Computer Science
City University of Hong Kong
Shussain7-c@my.cityu.edu.hk

Jacky Keung
Department of Computer Science
City University of Hong Kong
Jacky.keung@cityu.edu.hk

Arif Ali Khan
Department of Computer Science
City University of Hong Kong
Aliakhan2-c@my.cityu.edu.hk

ABSTRACT

Context: In the plethora of studies, it has been empirically investigated that the incidence of design pattern instances can be considered as an indicator to elaborate the software design. The developers, who have more concern with design quality, are interested to know the effect of use intensity of design patterns on the system level design quality attributes. **Goal:** The objective of our study is to empirically investigate the effect of the frequent use of the Gang-of-Four (GoF) design patterns on the design quality attributes. **Method:** We perform a case study which includes three analyses in order to investigate, 1) the existence of a correlation between design pattern usage and design quality attributes, 2) the confounding effect of system size (number of classes) on the correlation, and 3) how the change in number of employed design pattern instances affects the design quality in the subsequent releases of a system. **Results:** The result of this study suggests that the reusability, flexibility and understandability have a significant relationship with the employed instances of Template, Adapter-Command, Singleton and State-Strategy design patterns, however, it is affected by the confounding effect of system size. Subsequently, in the subsequent releases of an open source project named velocity, we observed the use intensity of Singleton, Adapter-Command, and State-Strategy design patterns can improve the design quality in term of reusability and flexibility attributes.

CCS Concepts

• **General and Reference→ Cross-computing tools and techniques→***Empirical studies; Design; Metrics;*

Keywords

Design Patterns; Quality; Spearman Correlation; Reusability; Understandability; Flexibility; Extendibility; Effectiveness.

1. INTRODUCTION

The structure of object-oriented software includes a set of design entities named classes, methods, and packages besides their inheritance and composition features. In software engineering, the term pattern is used in different software development phases (such as in requirement and design phase) and with different application domains (such as enterprise applications) in order to refer a solution of a commonly occurring problem. In the domain of object-oriented software development, Gamma, Helm, Johnson, and Vlissides (also known as Gang-of-Four or GoF) introduce a catalog of 23 design patterns that describes simple and well-designed solutions to specific problems in object-oriented design [1]. The usage of design patterns can help to provide re-useable solutions and improve the system level quality attributes such as flexibility, understandability and reusability. In the preface of their GoF design patterns, Gamma et al. describes that "Our design pattern capture many of the structures that result of refactoring. Using these patterns early in the life of a design prevents later refactoring. Design patterns, thus provide targets for your refactoring" [1]. On the base of the close relationship between patterns and evolutionary design, J. Kerievsky [2] leads to the concept of refactoring to patterns with benefits and liabilities, and clarify the relationship between design patterns and early refactoring in a comprehensive and descriptive way.

In agile software development, the fundamental structure of software artifacts is highly provoked to its evolution with the passage of time through adding new features or modifying the existing functionality. Subsequently, it affects the system level quality attributes such as reusability, maintainability, understanding [3, 4]. There are two common practical approaches to address decline quality of software systems. The first approach is software refactoring, which is considered as a precise way to improve the design quality of a system by changing its internal structure without any change in its external behavior [5, 6], and the second approach is the employment of design patterns. Moreover, the occurrence of GoF design pattern improved the design properties of a system such as complexity, cohesion, and coupling, however, it is highly related to the confounding effect of system size in terms of the number of classes [7].

The aim of our study is to perform a case study which includes three analyses in order to empirically investigate the effect of usage of design patterns on the system level design quality attributes for the agile developers who have more concerns with design quality. Moreover, we also investigate the two factors, which affect the correlation (between employed design pattern instances and design quality attributes) that is the confounding effect of system size (number of classes) and change effect in subsequent releases of a system.

In this case study, the cases and unit of analysis are the Eclipse Java-based open source software projects which are retrieved from the Qualitas.class corpus in the compiled form [8]. In order to achieve our research objective and respond to our research questions, we perform data analysis using Spearman

978-1-5386-0592-9/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS.2017.37

correlation, descriptive statistics, non-parametric tests for hypothesis testing, 2-D and 3-D area charts. Though numerous studies have reported the relationship of employed design patterns and quality attribute [3, 4, 7, 9]. However, in this study, we use the QMOOD model since it has been thoroughly validated and used in many empirical studies to evaluate the system level design quality attributes [10], such as Reusability, Flexibility, understandability, Functionality, Effectiveness and Extendibility.

The rest of the paper is structured in the six sections: In section 2, we present the related work which highlights the importance and motivation of our research objective. In section 3, we provide a brief introduction of the hierarchical QMOOD model to discuss and present the linear equations for quantification of system level design quality attributes. In section 4, we present the structure of proposed case study, which includes subsections of research objective, research questions, cases and unit of analysis, used tools, and data analysis process. In section 5, we present and discuss the results in order to respond our research questions. In section 6, we present threats to validity. Finally, in section 7, we present the conclusion and future work.

2. RELATED WORK

Recently, in their study, A. Ampatzoglou et al [1] reported the interest of the software engineering research community for Gang-of-Four (GoF) design patterns in both academia and industry. In their mapping study of more than 130 scientific papers, they reported the effect of GOF design patterns on the software quality attributes remain more important research topic as compared to pattern formulations and pattern detection. Similarly, in their mapping study, C. Zhang and D. Budgen [12] also reported the usefulness of patterns in providing a framework for the maintenance and recommend that researchers should use case studies that's focused on the key patterns in order to identify the impact of their use on the quality attributes. M. Vokac et al [9] replicate the experimental study of L. Ratchet et al [13], and investigated the usefulness of design patterns with respect to maintenance in a design program, even though design problem is simpler than that solve by a pattern. Moreover, M. Vokac et al used real programming environment instead of pen and paper. In both studies, the authors compare the maintainability of systems with and without design patterns and use the same questionnaire to collect the data from professionals. Moreover, the authors considered the Abstract Factory, Composite, Observer, Visitor, and Decorator GoF design patterns, and reported different results especially in the case of Visitor and Observer patterns. Finally, the experimental results of both studies suggest that the impact of design patterns is either harmful or beneficial with respect to maintenance. Moreover, the applicability of GoF design pattern depends on decision provoked by designers through their common sense.

S. Jeanmart [14] perform an experiment to investigate the modifiability effort and understandability of Visitor design pattern instances. In their experimental study, the author used student participants as subject and conduct experiment with three open source projects as object. Each project includes both canonical and non-canonical representation of Visitor patterns. They conclude that the effort needed to modify the tasks can be reduced if the subjects have a good understanding of UML notations and canonical representation of the visitor pattern is used.

A. Ampatzoglou et al [4] perform a multi-project case study to investigate that which one is most reusable unit that is a class, a pattern or a package for the component based open source software projects. Subsequently, the authors include 23000 classes in a study that could be reused as pattern based components. Moreover, for each case, the authors investigate four alternatives, namely, a) reuse the class, b) reuses the pattern that a class belongs to, c) reuse the package that the class is included, and d) reuse all packages that include at least one class which participate in the pattern. Finally, the authors concluded that alternative reuse of the design pattern offer the optimal selection option as compared to other alternatives. Similarly, in their another study, A. Ampatzoglou et al [3] empirically investigate the effect of the GoF design patterns on the stability of classes in object-oriented applications. They conduct a multi-project case study with 65000 open-source Java classes and explore the probability of change in a class due to the propagation of changes occurs in other classes. The results of their study suggest that classes that play the single role in a GOF design pattern are more stable than those classes that play more than one role in GoF design pattern occurrences. Moreover, the results of A. Ampatzoglou et al [3] also suggest that different GoF design patterns provide the different level of stability of the classes that participate in them. Finally, they reported that the classes that play the aggregate role are less stable than classes that play the component role.

In a recent study [15], M. O. Elish and M. A, Mohammed performs an empirical and comparative study to investigate the use of design patterns and its impact on software quality. The aim of this study is to quantitatively measure and compare the fault density in design patterns in the object-oriented system at design, category, motif and role level. In comparison, of participants and non-participants classes, the author observes no clear tendency for the difference in fault density between participant and non-participant classes in a design motif. Similarly, in comparison of the fault density of participant classes across the different categories of design motif, the author observed that the classes which are participating in structural design motif have less fault density than the other categories. Finally, in comparison of the fault density of participant classes across the design motif, the authors observe that the design patterns such as Builder, Adaptor, Composite, Decorator, and Factory Method show significant differences. Recently, P. Sfetsos et al. [16] investigate and compare the effectiveness of employed design pattern instances in the design quality of software libraries and standalone application. Though, these studies have empirically investigated and reported the relationship between GoF design patterns and design quality, however, they cannot report the effect of

employed instances of the GoF design patterns on the design quality attributes with respect to confounding effect of system size and changes in incidence of design patterns instances in the subsequent releases of a system. Consequently, we present a case study to empirically investigate the correlation between the employed instances of the GoF design patterns and system level design quality attributes, and also investigate the factors which can affect the correlation, such as the confounding effect of system size and change in design pattern instances during the evolution of a system.

3. BRIEF INTRODUCTION OF QMOOD MODEL

J. Bansiya and C. G. Davis [10] introduces a four layer based hierarchical hierarchical Quality Model for Object-Oriented Design (QMOOD) to assess the object-oriented design quality attributes thoroughly to overcome the vague and multifaceted concept of software quality. In their study, J. Bansiya and C. G. Davis proposed a set of linear equations to predict and quantify the system level design quality attributes. The aim of QMOOD model is to relate the design properties (such as encapsulation, complexity, modularity, cohesion and coupling) and the system level design quality attributes (such as flexibility, understanding, effectiveness and reusability). Moreover, QMOOD model can quantify the quality in an elegant way and measure the software product quality using a set of quality attributes, characteristics, and metrics. The levels of hierarchical QMOOD are shown in Figure-1.

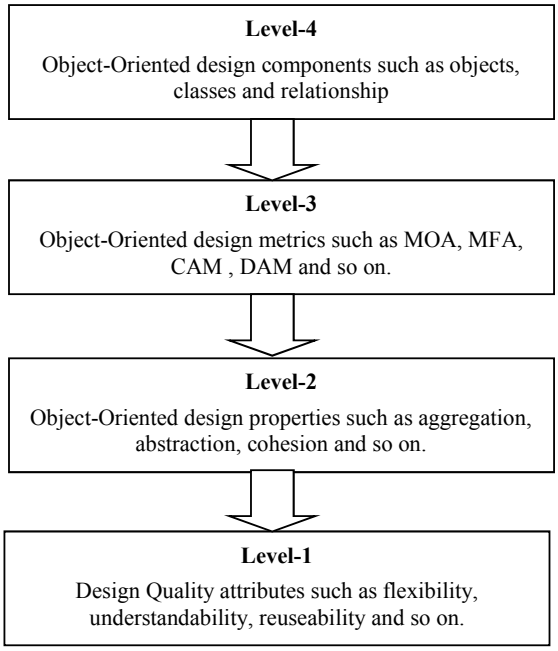


Figure 1. Layers of QMOOD Model

The linear equations of the QMOOD model for Reusability, Flexibility, Understandability, Functionality, Extendibility, and Effectiveness are shown from equation 1 to equation 6 respectively.

$$\text{Reusability} = -0.25 * \text{Coupling} + 0.25 * \text{Cohession} + 0.5 * \text{Messaging} + 0.5 * \text{Design Size} \quad (1)$$

$$\text{Flexibility} = 0.25 * \text{Encapsulation} - 0.25 * \text{Coupling} + 0.5 * \text{Composition} + 0.5 * \text{Polymorphism} \quad (2)$$

$$\text{Understandability} = -0.33 * \text{Abstraction} + 0.33 * \text{Cohession} - 0.33 * \text{Coupling} + 0.33 * \text{Encapsulation} - 0.33 * \text{Design Size} - 0.33 * \text{Complexity} - 0.33 * \text{Polymorphism} \quad (3)$$

$$\text{Functionality} = 0.12 * \text{Cohession} + 0.22 * \text{Polymorphism} + 0.22 \text{ Messaging} + 0.22 * \text{Design Size} + 0.22 * \text{Hierarchies} \quad (4)$$

$$\text{Extendebility} = 0.5 * \text{Abstraction} - 0.5 * \text{Coupling} + 0.5 * \text{Inheritance} + 0.5 * \text{Polymorphism} \quad (5)$$

$$\text{Effectiveness} = 0.2 * \text{Abstraction} + 0.2 * \text{Encapsulation} + 0.2 * \text{Composition} + 0.2 * \text{Inheritance} + 0.2 * \text{Polymorphism} \quad (6)$$

4. STRUCTURE OF CASE STUDY

In this paper, 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 [17]. We present this case study, according to rules suggested by P. Runeson et al [17]. In the domain of software engineering, P. Runeson et al present the design templates of two case-studies that are holistic and embedded case study, which are further classified as single-case and multiple-case study. According to P. 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 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 three types of analysis in order to achieve our research objective. Moreover, we perform these analyses with 51 open source project and nine subsequent releases of a system named velocity. We used the Qualitas.Class corpus to retrieve the compiled Eclipse Java open source software projects [8]. In each analysis of case study, we considered open source software projects as cases and unit of analysis. The results of the case study will aid the agile developers in order to evaluate the effectiveness of employed instances of the GoF design patterns in open source applications. Moreover, agile developers can also analyze and compare the effectiveness of the employed instance of the GoF design patterns across the open source applications with respect to system size (number of classes) and subsequent releases of a system.

The structure of this case study is described via the following subsections of research objective and research questions, data collection and used tools, and data analysis process.

4.1 Research Objectives and Research Questions (RQs)

The primary objective of our study is to empirically investigate the effect of use intensity of GoF design pattern

instances on the system level design quality attributes, and it's based on the achievement of three sub-objectives through the different analysis. The first sub-objective is to empirically investigate the existence of the correlation between the use intensity of the GoF design patterns and system level quality attributes. Through the statement of our first sub-objective, we extract and formulate our first research question.

RQ-1. Is there any correlation between use intensity of GoF design pattern and system level design quality attributes?

Subsequently, in their study A. Ampatzoglou and A. Chatralampidou [7] reported that the GoF design pattern occurrence can improve the design properties such as complexity, cohesion, and coupling, however, it is highly provoked with system size in terms of the number of classes or line of code. The second sub-objective of our study is to investigate the confounding effect of system size on the correlation between use intensity of the GoF design patterns and design quality attributes. Through the statement of our second sub-objective, we extract and formulate our second research question.

RQ-2. What is the impact of the confounding effect of system size (# of classes) on the correlation between use intensity of the GoF design patterns and the design quality?

Moreover, in the evolution process of software systems, it has been empirically validated that employed design pattern instances are changed and causes the modification to the design components (such as method or classes), either as participant or non-participant in the design pattern. Moreover, it can also affect the system level design quality. In our study, we consider the QMOOD model based system level design quality attributes, which are calculated through linear equations.

Table 1. Structure of Dataset

Variable(s)	Alias	Description
V1	Project Name	We used this variable to record the name of project as a case or unit of analysis.
V2	Project Size	This Variable refers to a number of classes exist in a project.
V3-V14	Use Intensity of Design Patterns	We used these variables to record the use intensity of each type of design patterns, including Proxy2 as a variation of Proxy pattern. The variable V2 to V13 refer to Singleton, Factory Method, Prototype, Adapter-command, Observer, State-Strategy, Template Method, Visitor, Composite, Decorator, Proxy and its variation Proxy2 respectively. The values of these variables are normalized with system size and the employed design pattern instances, that is, each value is calculated as the fraction of the number of classes (system/project size) divided by the number of instances of corresponding design pattern. The low value of variable refers to the intensive use of the corresponding pattern in a project.
V15-V24	Metric values for Design Paroperties	We used these variables to record the metric values which help to quantify the design properties (such as cohesion, coupling, abstraction and so on) and design quality attributes. The value of each variable (from V15 to V24) of a case (an open source project) refers to an Average Value (AV) and calculated as $AV_{i,k} = \frac{\sum_{j=1}^{N} \text{Value\_of\_Case}_i\text{\_for\_Class}_j}{N}$ $\forall$ i=1 to n and k=15 to 24 The value of $AV_{i,k}$ refers to the average value of variable k in case I, such that i ε n, where n refer to the total number of cases in the dataset. The term N refers to the system size that the total number of classes for each case i.
V25-V30	Design Quality Attributes	We used these variable to record the quantified values of system level quality attributes such as Understandability, Reusability, Functionality, Flexibility, Effectiveness and Extendibility. The value of each variable (from V25 to V30) of a case (an open source project) refers to the Average Value (AV) and calculated as $AV_{i,k} = \frac{\sum_{j=1}^{N} \text{Value\_of\_Case}_i\text{\_for\_Class}_j}{N}$ $\forall$ i=1 to n and k=25 to 30 The value of $AV_{i,k}$ refers to the average value of variable k for case i, such that i ε n, where n refer to the total number of cases in the dataset. The term N refers to the system size that the total number of classes for each case.

Subsequently, each equation includes the low-level metric values of the corresponding design properties, such as MOA (Measure of Aggregation) used to present the design property called Composition. The change in the incidence of design pattern instances in any release of a software system can also affect the design quality in its subsequent releases. The third

sub-objective of our study is to investigate the change in the correlation between use intensity of the GoF design patterns and design quality attributes in subsequent releases of a system. Through the statement of our third sub-objective, we extract and formulate our third research question.

RQ-3. What is the effect of a change in the employed instances of a GoF pattern on design quality in the subsequent releases of a system?

4.2 Data Collection and Used Tools

We employed two different tools to collect data from the compiled Eclipse Java open source projects retrieved from Qualitas.class corpus[1]. The first tool is SSA (Similarity Scoring Algorithm) introduce by Tsantalis[2] et al [18] and it is based on a similarity scoring algorithm. Moreover, the SSA tool is capable to identified the occurrences of the GoF design patterns exist either in standard or in variation form. The second tool is CKJM-extend[3] which is extended version of CKJM (Chidamber and Kemerer Java Metrics) tool [19]. Moreover, this tool helps to measure the metrics of different suites such as Chidamber and Kemerer (CK), QMOOD, Tang et al., and Martin suites. The values of these metrics can be used to quantify the design properties and system level quality attributes. In order to perform their function, both tools need the compiled form (byte code) of open source project and can be downloaded from the web[2,3]. The structure of a created dataset is described in Table 1.

4.3 Data Analysis

We performed three analyses using descriptive statistics, which include mean values and Spearman correlation, and graphs (2D line and 3D area charts) to present monotonic relationship and effect of employed instances of design patterns on the design quality attributes. Moreover, we also perform hypothesis testing using non-parametric Friedman test in order to evaluate the design quality differences among project's groups with respect to system size.

5. RESULTS AND DISCUSSION

In this section, we present the results of our case study, which are organized to respond our research questions.

5.1 Respond to RQ-1.

In order to respond to our first research question (RQ-1), we perform non-parametric (distribution-free) Spearman's correlation to investigate the monotonic relationship between use intensity of design patterns and design quality attributes. The Spearman's correlation (Corr) and significant value p for GoF design pattern and corresponding design quality attributes are shown in Table 2. Moreover, each value in Table 2 is represented as Corr (p).

Table 2. Spearman Correlation between GoF design pattern and Quality attributes

Design Patterns	Reusability	Flexibility	Understandability	Functionality	Extendibility	Effectiveness
Singleton	0.60(0.00)	0.33(0.03)	-0.60(0.00)	0.58(0.00)	-0.36(0.03)	0.42(0.00)
Factory Method	0.60(0.00)	0.32(0.01)	-0.60(0.00)	0.60(0.00)	-0.34(0.03)	0.37(0.00)
Prototype	0.35(0.01)	-0.01 (0.93)	-0.35(0.01)	0.32(0.02)	-0.23(0.10)	0.22(0.11)
Adapter-Command	0.63(0.00)	0.29(0.03)	-0.64(0.00)	0.65(0.00)	-0.33(0.01)	0.51(0.00)
Observer	0.68(0.00)	0.37(0.02)	-0.68(0.00)	0.67(0.00)	-0.31(0.02)	0.37(0.00)
State-strategy	0.6(0.00)	0.32(0.01)	-0.6(0.00)	0.64(0.00)	-0.34(0.01)	0.47(0.00)
Template Method	0.62(0.00)	0.17(0.24)	-0.62(0.00)	0.62(0.00)	-0.15(0.29)	0.29(0.03)
Visitor	0.07(0.61)	0.09(0.51)	-0.07(0.61)	0.05(0.75)	-0.07(0.63)	0.12(0.39)
Composite	0.23(0.09)	0.09(0.51)	-0.24(0.09)	0.27(0.05)	-0.21(0.14)	0.34(0.01)
Decorator	0.43(0.00)	0.12(0.40)	-0.43(0.00)	0.48(0.00)	-0.23(0.10)	0.37(0.00)
Proxy	0.48(0.00)	0.09(0.00)	-0.48(0.00)	0.48(0.00)	-0.50(0.00)	0.49(0.00)
Proxy2	-0.03(0.84)	0.08(0.58)	0.03(0.85)	-0.01(0.92)	-0.22(0.12)	0.22(0.12)

In Table 2, the high value with a negative sign also presents the significant correlation between number of employed instances of design patterns and quality attributes. The negative sign is due to the fact that quality attributes are quantified using equations (from equation 1 to equation 6) and use intensity of design patterns is acquired by the fraction of a number of classes and number of instances of corresponding design pattern. The main consequences which are extracted from the results of Table 2 are.

- We observe the significant correlation (at the level $p<0.05$) between the number of employed instances of Singleton, Factory Method, Adapter-Command, Observer, State-strategy and Proxy design patterns and the system level quality attributes. Moreover, these design patterns are heavily used and their aggregate effect is evident at the system level.
- We observed that there is no significant correlation of employed instances of proxy2 (a variant of proxy) on design quality attributes, it might be that it cannot be

[1] http://java.labsoft.dcc.ufmg.br/qualitas.class/download.html
[2] http://users.encs.concordia.ca/~nikolaos/pattern_detection.html
[3] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/

detected in many applications due to lack of the developers interest.

- We also observe the differences in the significance effect of the number of employed instances of rest of design patterns on the design quality attributes, such as the number of employed instances of the Template Method design pattern have a significant effect on all design quality attributes except the extendibility and flexibility.

Table 3. Group's information

Groups	# of Cases	System Size (# of classes)
Group-1	17	<100
Group-2	17	100-1000
Group-3	17	>1000

From the findings of Table 2, we can conclude that there is an evidence in the support of existence of a correlation between use intensity of design patterns and system level quality attributes, however, their correlation is highly related with the number of employed instance of design patterns and the values of metrics which are used to quantify the design quality attributes

Table 4. Descriptive Statistics of Design Quality Attributes

Quality Attributes	Mean Values		
	Group-1	Group-2	Group-3
Reusability	31.97	161.64	517.26
Flexibility	0.25	0.56	0.49
Understandability	-25.51	-112.30	-348.08
Functionality	22.25	116.36	374.37
Extendibility	-1.74	-1.85	-2.37
Effectiveness	0.78	1.09	1.25

Table 5. Results of Friedman's test for Quality Attributes

Quality Attributes	Null Hypothesis (HP0)	Chi-squared	p-value	HP0 Decision
Reusability	The distribution of Reusability is same across all groups	34	0.00	Rejected
Flexibility	The distribution of Flexibility is same across all groups	8.94	0.01	Rejected
Understanding	The distribution of understandability is same across all groups	34	0.00	Rejected
Functionality	The distribution of Functionality is same across all groups	34	0.00	Rejected
Extendibility	The distribution of Extendebility is same across all groups	8.74	0.01	Rejected
Effectiveness	The distribution of Effectiveness is same across all groups	10.70	0.004	Rejected

5.2 Respond to RQ-2.

In order to respond our RQ-2, we perform an analysis with respect to system size and divide the cases (that is open source projects) into three groups that are Group-1, Group-2, and Group-3. For each group, we create a new dataset of the same structure defined in Table 1. The system size and number of cases of each group is shown in Table 3. Moreover, we investigate the differences among three groups with respect to design quality, use intensity of design patterns and the effect of GoF design pattern instances on the design quality attributes.

A. Differences among groups in term of design quality

In order to evaluate the differences among groups in term of design quality, we perform this analysis in two steps. In the first step, we find the mean descriptive statistical value of design quality attributes for Group-1, Group-2, and Group-3 respectively, which are shown in Table 4. Subsequently, in the second step, we perform a non-parametric Friedman's test in order to investigate the differences between mean values of design quality attributes across the groups. The null hypothesis (HP0) for each design quality attribute and its decision is shown in Table 5.

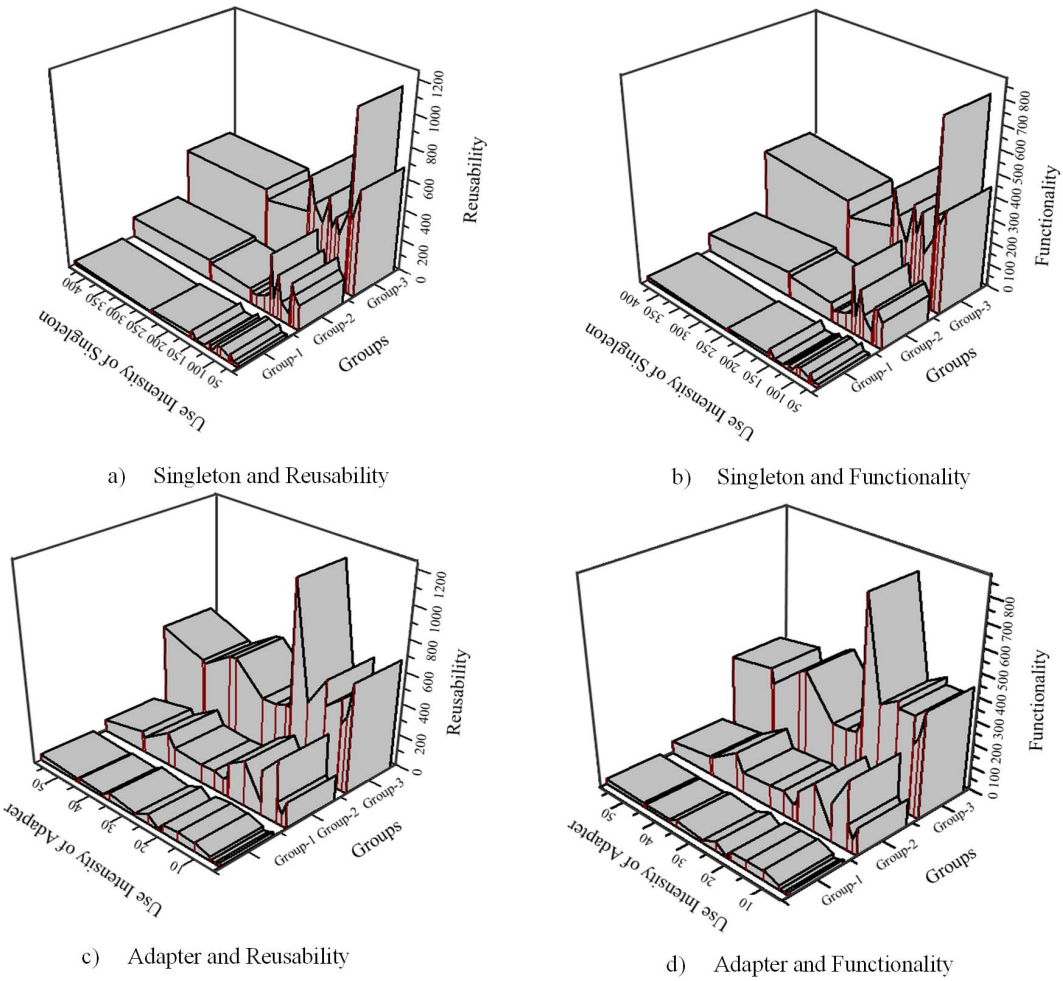
Table 6. Descriptive Statistics of GoF Design Patterns

Design Patterns	Mean Values		
	Group-1	Group-2	Group-3
Singleton	21.75	80.45	195.85
Factory Method	10.24	93.72	239.36
Prototype	10.89	31.62	253.79
Adapter-Command	22.58	23.41	25.36
Observer	5.59	89.56	427.69
State-strategy	19.79	23.85	13.43
Template Method	25.15	72.28	150.03
Visitor	1.12	2.19	9.01
Composite	0.00	26.87	241.72
Decorator	0.43	73.80	231.21
Proxy	5.59	99.79	311.99
Proxy2	0.23	22.66	154.14

We can conclude from the findings (presented in Table 4 and Table 5) that there is evidence of differences in design quality attributes of Group-1, Group-2, and Group-3 projects.

Table 7. Results of Friedman's test for GoF Design Patterns

Design Pattern	Null Hypothesis (HP_0)	Chi-Sq	p-value	HP_0 Decision
Singleton	The use intensity of Singleton is same across all groups	13.06	0.00	Rejected
Factory Method	The use intensity of Factory Method is same across all groups	18.37	0.00	Rejected
Prototype	The use intensity of Prototype is same across all groups	0.89	0.01	Rejected
Adapter	The use intensity of Adapter is same across all groups	12.23	0.03	Rejected
Observer	The use intensity of Observer is same across all groups	18.49	0.00	Rejected
State-strategy	The use intensity of State-strategy is same across all groups	12.24	0.03	Rejected
Template Method	The use intensity of Template Method is same across all groups	14.36	0.00	Rejected
Visitor	The use intensity of Visitor is same across all groups	0.00	0.10	Accepted
Composite	The use intensity of Composite is same across all groups	7.84	0.03	Rejected
Decorator	The use intensity of Decorator is same across all groups	23.05	0.00	Rejected
Proxy	The use intensity of Singleton is same across all groups	15.26	0.00	Rejected
Proxy2	The use intensity of Singleton is same across all groups	1.75	0.42	Accepted



a) Singleton and Reusability

b) Singleton and Functionality

c) Adapter and Reusability

d) Adapter and Functionality

Figure 2. 3-D Chart Areas

Consequently, we can also conclude that distribution of design quality is highly related to the confounding effect of system size.

B. Differences among groups in term of use intensity of design patterns

In this analysis, we investigate that difference in quality distribution among Group-1, Group-2, and Group-3 projects can be attributed to use intensity of the GoF design pattern if we find differences in their use intensity. Moreover, we perform this analysis in two steps. In the first step, we find the mean descriptive statistical value of employed design pattern instances for Group-1, Group-2 and Group-3 respectively, which are shown in Table 6.

Table 8. Spearman's Correlation across the Groups

Design Pattern :: Quality Attribute	Group	Corr. Coeff (ρ)	Sig (p)
Singleton :: Reusability	Group-1	0.61	0.01
	Group-2	0.47	0.02
	Group-3	0.31	0.22
Singleton :: Flexibility	Group-1	0.61	0.01
	Group-2	0.04	0.89
	Group-3	0.13	0.61
Singleton :: Understandability	Group-1	-0.64	0.01
	Group-2	-0.48	0.01
	Group-3	-0.42	0.02
Singleton :: Functionality	Group-1	0.54	0.02
	Group-2	0.31	0.23
	Group-3	0.25	0.33
Adapter-Command :: Reusability	Group-1	0.68	0.00
	Group-2	0.44	0.01
	Group-3	0.09	0.73
Adapter-Command :: Flexibility	Group-1	0.47	0.02
	Group-2	0.38	0.04
	Group-3	0.54	0.02
Adapter-Command :: Understandability	Group-1	-0.71	0.00
	Group-2	-0.45	0.01
	Group-3	-0.10	0.71
Adapter-Command :: Functionality	Group-1	0.73	0.00
	Group-2	0.42	0.09
	Group-3	0.33	0.06
State-Strategy :: Reusability	Group-1	0.51	0.03
	Group-2	0.12	0.63
	Group-3	0.01	0.98
State-Strategy :: Flexibility	Group-1	0.44	0.07
	Group-2	0.39	0.02
	Group-3	0.38	0.02
State-Strategy :: Understandability	Group-1	-0.54	0.02
	Group-2	-0.15	0.57
	Group-3	0.00	0.99
State-Strategy :: Functionality	Group-1	0.55	0.02
	Group-2	0.12	0.63
	Group-3	0.11	0.68
Template :: Reusability	Group-1	0.61	0.01
	Group-2	0.45	0.06
	Group-3	0.11	0.68
Template :: Flexibility	Group-1	0.46	0.06
	Group-2	0.11	0.66
	Group-3	0.11	0.68
Template :: Understandability	Group-1	-0.61	0.01
	Group-2	-0.45	0.06
	Group-3	-0.11	0.68
Template :: Functionality	Group-1	0.57	0.01
	Group-2	0.45	0.07
	Group-3	0.12	0.65

Subsequently, in the second step, we perform a non-parametric Friedman's test in order to investigate the differences between mean values employed design pattern instances across the groups. The null hypothesis (HP0) for each design pattern and its decision is shown in Table 7. The findings of Table 6 and Table 7 suggest the differences among Group-1, Group-2, and Group-3 in term of use intensity of design patterns, such as in Table 6, we can observe the incidence of employed design pattern instances is high in Group-3 as compared to Group-1 and Group-2, which refer to the confounding effect of system size. Moreover, The hypothesis testing results (shown in Table 7) of each design pattern suggest us that the design quality distribution across the groups can be attributed to the use intensity of design patterns.

C. Differences in the effect of design patterns on design quality in Groups

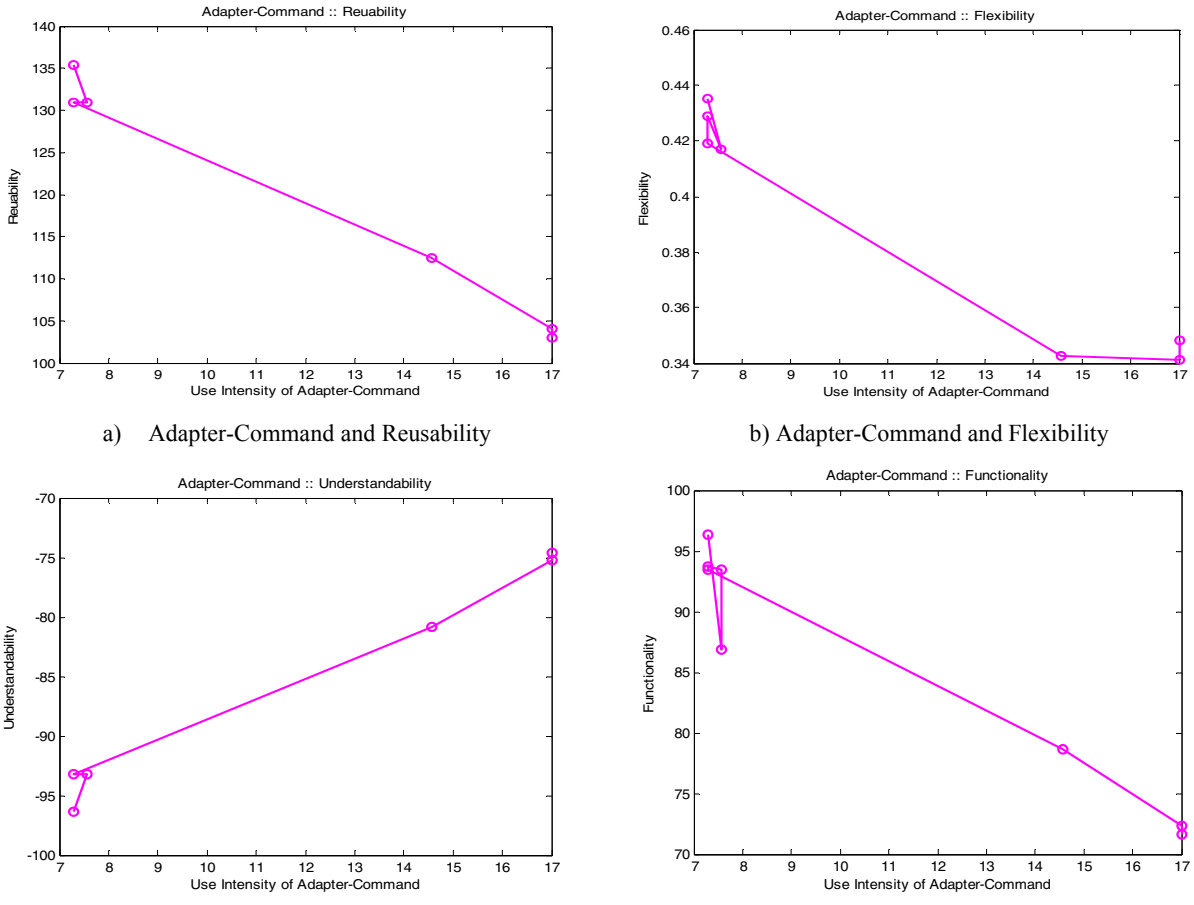
In section 5.1, we have investigated that design patterns such as Singleton, Factory Method, Adapter-Command and Template have statistically strong significant correlation with design quality attributes (also shown in table 2). Subsequently, in this analysis, we further investigate that how system size influences the significant strength of correlation between the number of employed instances of these (such as Singleton, Factory Method, Adapter-Command, and Template) design patterns and design quality attribute. Again, we perform non-parametric (distribution-free) Spearman's correlation to measure the strength of the relationship between use intensity of design patterns and design quality attributes in each group. The detailed results of this analysis are shown in table 8.

From these results, we can conclude that in the case of group-1 projects, employed instances of Singleton, Adapter-Command, State-Strategy and Template design patterns have a significant effect on design quality attributes, while in the case of group-2 and group-3 projects, we can observe this effect at different significant levels. For example, the effect of employed instances of Adapter-Command on the flexibility can be observed as ρ=0. 38 (group-2) and ρ=0.54 (group-3) at significant level p=0.04 and p=0.02 respectively. Moreover, we also explore interesting findings to differentiate the effect of the number of employed design pattern instances on the quality attributes with respect to system size through the 3-D area chart plot. The x-axis of each plot presents the use intensity (with a fraction of a number of classes and employed instances) of a design pattern, y-axis presents the mean values of quality attributes, and overall plot to present the effects across the groups. The value of design quality attribute (y-axis) can be interpreted with respect to the corresponding use intensity of a design pattern (x-axis). For example, drop lines on XY faces of Figure 2(c) present the influence of use intensity of Adapter-Command pattern on the Reusability. Moreover, we can also observe an increase in Reuseability

with frequent use of Adapter-Command pattern (low use intensity value that is 50) for each group especially in group-3. Finally, from the results shown in table 4, Table 5, Table 6, Table 7, and Table 8, it can be concluded that system size has more concern in order to investigate the effect of use intensity of the GoF design patterns on the system level quality attributes.

5.3 Respond to RQ-3.

In agile development, there are two common practical approaches to address decline quality of software systems with the passage of time such as refactoring and employment of design patterns. In this analysis, we investigate the influence of a change in the number of employed design pattern instances on the system level design quality attributes in subsequent releases of a system. Moreover, in this analysis, we consider nine subsequent releases of an open source project named velocity and create a dataset with respect to the structure defined in Table 1. Subsequently, during analysis, we observe no change in the number of employed instances of the GoF design patterns except Adapter-Command, State-Strategy, and Template design patterns. We also observe the significant effect of use intensity of Adapter-Command, State-Strategy, and template design patterns on the design quality attributes through Spearman correlation, such as the use intensity of State-strategy has a significant correlation with quality attributes Reusbility (ρ=0. 83), Flexibility (ρ =-0.71), Understandability (ρ=0.83), Functionality (ρ=-0.82), Extendibility (ρ=0. 86) and Effectiveness (ρ=-0.77) at level $p<0.05$. Moreover, we plot a 2-D line chart in order to present the change in design quality with respect to changes in the employed design pattern instances in subsequent releases. For example, in Figure 3, we present the monotonic relationship between use intensity of Adapter-command pattern and the variations in the design quality attributes. The x-axis of plots (Figure 3) presents the use intensity value of design patterns (from a low value to high) and y-axis presents the quality attributes values. The low value of use intensity of a design pattern refers to the high frequency of employed instances. From Spearman correlation results and plots shown in Figure3, it can be suggested that the change in the employed instances of design patterns in a subsequent release of an application also affect its design quality attributes, such as improvement in reusability due to the frequent use of Adapter design pattern.



a) Adapter-Command and Reusability

b) Adapter-Command and Flexibility

c) Adapter-Command and Understandability

d) Adapter-Command and Functionality

Figure 3. Correlation between Use Intensity and Quality Attributes in Subsequent releases of Velocity

6. THREAT TO VALIDITY

In this section, we describe some internal and external threats to validity in order to present our case study. The first threat is related to the size of the dataset for empirical investigation of the effect of use intensity of design patterns on the quality attributes. We select 51 open source projects which are further classified into groups with respect to system size. The results could be altered by increasing the number of cases in the proposed case study. The second threat is about the selected design patterns, our results on 12 design patterns cannot be generalized to the rest of 23 GoF design patterns. The third threat is that the results and conclusion of our research are strongly depends on the QMOOD model, however, there are certain models for different quality attributes which can alter the results.

7. CONCLUSION

The promising results of the proposed study (in terms of the relationship between design pattern usage and quality) can aid the developers to monitor the systems and relevant activities in real-life context. In this paper, we present a holistic multiple-case study with three types of analysis, which can aid agile developers to evaluate the effectiveness of employed instances of the Gang-of-Four (GoF) design patterns on the design quality in the domain of open source applications. In the first analysis, we investigate the statistical significant correlation between the use intensity of the GoF design patterns and system level design quality attributes. Subsequently, in the second analysis, we investigate the confounding effect of system size on the correlation. In the third analysis, we investigate the effect of a change in the number of employed pattern instances on the design quality attributes in subsequent releases of an application. The cases and unit of analysis of the case study are 51 open source projects and nine subsequent releases of an open source project named Velocity. The main consequences of the proposed study are; 1) there is a significant relationship between design pattern usage and quality. For example, the employed instances of Adapter-Command, Singleton, State-Strategy and Template design patterns have a significant impact on the reusability, flexibility and understandability, 2) The system size also has the effect on the relationship. For example, the significant difference between the relationship of Adapter with Reusability and functionality across the groups, and 3) the usage of Adapter-Command and State-Strategy design patterns improve the design quality (in terms of reusability and flexibility) in the subsequent releases of velocity (an open source project). Subsequently, in the future, we will investigate the effect of the GoF design pattern on the design quality at class-level rather than system-level in order to provide more evidence in the support of our objectives.

8. ACKNOWLEDGEMENT

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No. 125113, 11200015 and 11214116), and the research funds of City University of Hong Kong (No. 7004683 and 7004474).

9. REFERENCES

[1] E. Gamma et al., Design Patterns : Elements of Reusable Object-Oriented Software. 1995, Boston: Addison-Wesley.

[2] J. Kerievsky., Refactoring to Patterns, Addison Wesley, 2004.

[3] A. Ampatzoglou et al., The Effect of GoF Design Patterns on Stability: A Case Study. IEEE Transactions on Software Engineering, 41(8): p. 781-802, 2015.

[4] A. Ampatzoglou, et al., An empirical investigation on the reusability of design patterns and software packages, The Journal of System and Software, 84(12): p. 2265-2283, 2011.

[5] M. Fowler, Refactoring Improving the Design of Existing Code, Addison Wesley, 1999.

[6] C.J. Neill, Leveraging object-orientation for real-time imaging systems. Real-Time Imaging, 9(6): p. 423-432, 2003.

[7] A. Ampatzoglou, and A. Chatzigeorgiou, Evaluation of object-oriented design patterns in game development, Information and Software Technology,49 (5), p. 445-454, 2007.

[8] R. Terra et al., Qualitas.class Corpus: A compiled version of the Qualitas Corpus," Software Engineering, Engineering Notes, 38(5), pp. 1–4, 2013.

[9] M. Vok et al., A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns; A Replication in a Real Programming Environment. Empirical Software Engineering, 2004. 9 (3): p. 149-195, 2004.

[10] J. Bansiya, and C. G. Davis, A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transaction on Software Engineering, 28(1): p. 4-17, 2002.

[11] A. Ampatzoglou et al, Research State of the art on GoF Design Patterns: A mapping study, Journal of System and Software, 86, p. 1945– 1964, 2013.

[12] C. Zhang, and D. Budgen, A survey of experienced user perceptions about software design patterns. Information and Software Technology, 2013. 55(5): p. 822-835, 2013.

[13] L. Prechelt et al, A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions. IEEE Transactions on Software Engineering, 27(12): p. 1134-1144, 2001.

[14] Jeanmart, S., et al. Impact of the visitor pattern on program comprehension and maintenance, Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009.

[15] M. O. Elish and M. A. Mohammed, Quantitative analysis of fault density in design patterns. Information and Software Technology, 66(C): p. 58-72, 2015.

[16] P. Sfetsos et al, A Comparative Study on the Effectiveness of Patterns in Software Libearies and Standalone Applications, 9th International Conference on the Quality of Information and Communication Technology, 2014.

[17] P. Runeson et al, Case Study Research in Software Engineering: Guidelines and Examples, John Wiley and Sons, 2012.
[18] N. Tsantalis et al, Design Pattern Detection using Similarity Scoring, IEEE Transaction on Software Engineering, 32(11), pp. 896-909, 2006.
[19] M. Jureczko and D. Spinellis, Using object-oriented design metrics to predict software defects, Monographs of system dependability, models and methodology of system dependability, Wroclaw, Poland, 2010, pp. 69-81. http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/