An empirical investigation on the impact of design pattern application on computer game defects

Apostolos Ampatzoglou
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
apamp@csd.auth.gr

Apostolos Kritikos
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
akritiko@csd.auth.gr

Elvira-Maria Arvanitou
Technological Education Institute
Department of Informatics
Thessaloniki, Greece
earvanit@it.teithe.gr

Antonis Gortzis
Technological Education Institute
Department of Informatics
Thessaloniki, Greece
gkortz@it.teithe.gr

Fragkiskos Chatziasimidis
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
fchatzia@csd.auth.gr

Ioannis Stamelos
Aristotle University of Thessaloniki
Department of Informatics
Thessaloniki, Greece
stamelos@csd.auth.gr

ABSTRACT
In this paper, we investigate the correlation between design pattern application and software defects. In order to achieve this goal we conducted an empirical study on java open source games. More specifically, we examined several successful open source games, identified the number of defects, the debugging rate and performed design pattern related measurements. The results of the study suggest that the overall number of design pattern instances is not correlated to defect frequency and debugging effectiveness. However, specific design patterns appear to have a significant impact on the number of reported bugs and debugging rate.

Categories and Subject Descriptors
D.2.10 [**Software Design]:** Methodologies

General Terms
Design

Keywords
Design patterns; software defects; empirical study; computer games

1. INTRODUCTION
In recent years, computer game design is a rapidly growing field of computer science [18]. Until the first half of the 90's, game applications were written from scratch in Assembly language and game developers did not aim at creating reusable code [19]. Later on, the concept of code reuse was introduced as a major breakthrough in game development because games started becoming more complex and the process of their production was much more time consuming. In [2], it is suggested that game development companies have altered game lifecycle and their project management process. More specifically, it is suggested that due to the reduction of development time, games may be delivered to market with bugs. For this reason, software patches have become a common practice, which deals with debugging, i.e. repairing errors. In order for this problem to be tackled, frameworks and game engines have been produced. A framework is a collection of components that can be widely reused and be integrated with others components [20 and 22]. Usually frameworks implement mechanisms that are parts of many games (e.g. such as input management, file management (texture, models, audio etc), 3D rendering etc.). Game engines are programs that give developers the opportunity to design game levels, handle player and oppositional behaviour, by handling scripting languages and powerful GUIs. Consequently, if frameworks and game engines are "well-structured", they can be maintained without marginal effort and be adapted so as to support a variety of game genres.

Design patterns are generic, reusable solutions to frequent problems in software design [6]. Patterns' purpose is to capture design knowledge in a form that can be easily reused. According to [6], object-oriented design patterns usually present relationships and interplay between classes or objects. In [6] the authors imply that design pattern employment strengthens software maintenance, flexibility and makes future adoptions, an easy task. In contrast to that, many studies propose that design pattern applications do not end up being beneficial as far as software quality is concerned [11]. Currently there are two approaches on the term of game design patterns, i.e. the use of patterns on game mechanics and the use of object-oriented design patterns in game programming.

In this paper, we examine the relationships among (1) design pattern application, (2) game defect frequency and (3) game debugging efficiency. In section 2, we present a review of the current state of the research on game development and design patterns. In section 3, we deal with the methodology of our study. In section 4, we present our results and in section 5, we discuss the research questions of our study. Finally, sections 6 and 7 supply a discussion about threats to validity, future research plans and conclusions.

MindTrek'11, September 28-30, 2011, Tampere, Finland

2. BACKGROUND INFORMATION

In this section, we present the results from a literature review focused on the effect of object-oriented design patterns to software quality attributes. In [11] the authors gathered and presented information on every GoF design pattern and several high level quality attributes, by conducting a survey on a group of professional software engineers. Additionally, in [16 and 23], professional software engineers participated in experiments in order to evaluate how several design patterns like abstract factory, composite, decorator, observer and visitor affect quality attributes such as maintainability, understandability and stability. Although the results show that design patterns are usually the best solution for common design problems, their application must be in accordance with the discretion of each software designer. In [7] the authors discuss how the application of patterns affects the system's change proneness and report their conclusions. In [9] the usability of abstract factory pattern in API design was investigated and the results were validated by professional programmers who participated in a controlled experiment.

Moreover, the maintainability and understandability of the visitor pattern is discussed by the author of [10]. In [17], the authors performed a case study on java open source software in order to explore the reusability and the modularity of the GoF design patterns. The results show that the reusability of code is adversely affected by patterns such as bridge, flyweight, interpreter, mediator, memento, singleton, state and visitor whereas patterns such as singleton, bridge, flyweight, memento and state lag with respect to modularity. In [8] the author discusses the stability of four structural patterns by thoroughly examining and qualitatively evaluating their class diagrams. In [24], the author investigates how the application of patterns such as abstract factory, decorator, observer, singleton and template method affects the defect frequency of systems. The results indicate that the application of abstract factory and template method decrease the system defect frequency, while extended use of the observer pattern tends to have adverse effects. Additionally the article examines whether the combination of patterns affects the stability of a system. In [4], authors utilize a formal approach in order to study the testability of mediator, observer and visitor pattern. The maintainability of the proxy design pattern is investigated in [13] through a case study on open source software, whereas in [25] the author investigates design patterns' understandability and the negative effects of their removal in industrial software.

Regarding computer game development, two studies [1 and 15] attempted to investigate how the application of object-oriented design patterns affects the quality of the derivative software. More specifically, in [1], the authors analyzed existing systems and examined how the application of patterns affects the game structure and maintainability. The analysis indicated that using patterns improves attributes such as complexity and coupling of the game and cohesion of the code, although affecting the project's size by increasing its lines of code. In [15], authors attempted to create a pattern-based game and suggested that using design patterns is an optimal solution to achieve decoupling and abstractions in a game. Authors in [14] describe how mechanisms of virtual reality systems can be implemented in game development. Finally, in [5] the authors report that in large project development like computer games, difficulties appear in the collaboration among staff with different expertise. The study suggests that patterns should be used as a tool in order to deal with these difficulties effectively and efficiently.

3. METHODOLOGY

According to the authors of [26], three major empirical investigation approaches exist, surveys, case studies and experiments. Considering the nature, the subject of our research and the plethora of available open-source projects we believe that a case study is the most suitable for our research needs.

In this section of the paper we describe the methodology of our case study. The case study of our research was based on the guidelines described in [12]. According to [12] the steps for conducting a case study include:

(a) Define hypothesis
(b) Select projects
(c) Method of comparison selection
(d) Minimization of confounding factors
(e) Planning the case study
(f) Monitoring the case study and
(g) Analyze and report the results

The hypotheses, i.e. step (a), are defined in section 3.1. Steps (b) and (d) which deal with project selection protocol and minimizing confounding factors are presented in section 3.2, accompanied with step (e). The methods used in analyzing the data, i.e. step (c), is presented in section 3.3, step (f) as it is described in [12] is discussed in section 6. Finally, concerning step (g), we report the results on section 4 and discuss them in section 5.

3.1 Research Questions

In this section of the paper we state the research questions that are investigated in our study.

RQ1: Is design pattern usage related to number of defects in java open source games?

RQ2: Is design pattern usage related to successful debugging activities in java open source games?

3.2 Case Study Plan

According to [3], in order to produce a solid methodology for an empirical validation method, a study plan should be thoroughly designed. In this case study the plan involved a five step procedure:

1. identify a number of projects that fulfil certain selection criteria in the domain of computer games
2. perform pattern detection for every selected project (the detected patterns are Abstract Factory, Singleton, Composite, Adapter, Observer, State, Strategy, Template Method, Decorator, Prototype and Proxy).
3. perform bug tracking analysis for every selected project in order to identify the number of bugs open and bugs fixed
4. tabulate results
5. analyze data with respect to the research questions

From the available OSS games we have selected projects that fulfilled the following criteria:

1. software written in java, due to limitations of a pattern detection tool [21]
2. software that provides binary code, due to limitations of the pattern detection tool
3. software that had more than 10 reported bugs for each of its releases

A possible confounding factor of this study is that both design pattern use intensity and defect frequency are correlated to the size of the software. So, larger programs are expected to have more bugs and more pattern instances. Thus, results which

suggest that design pattern instances are positively correlated to number of defects should be cautiously adopted. Additionally, another factor that should be considered during the interpretation of the results is the degree of pattern knowledge of open-source game developers. It is believed that more experienced developers are more probable to use more complex patterns and less experienced developers to use rather simpler, such as State, Strategy and Adapter. Additionally, we believe that more experienced developers are more probable to write bug-free code.

3.3 Data Analysis Methods

The resulted dataset after design pattern detection and bug tracking analysis included only numerical data. However, some techniques that were employed during data analysis need categorical or binary variables. Thus, certain data transformations have taken place. On the completion of the pre-processing phase each project was characterized by 64 variables:

1. name
2. version
3. defect frequency (bugs open)
4. bugs fixed
5. debugging efficiency (i.e. bugs fixed divided by bugs opened)
6. number of classes
7. three variables for each pattern (number of pattern instances, count of classes that participate in the pattern. i.e. pattern participants, percentage of project classes that participate in every instance of the pattern). That is 33 variables
8. two categorical variables for each pattern. That is 22 variables
9. overall pattern participants percentage. That is the number of classes' percent of project classes that are employed in at least one pattern
10. two categorical variables for characterizing the project according to the total number of pattern participating classes.

The analysis phase of our study has employed statistical methods, such as descriptive statistics, independent sample t-tests and boxplots. The statistical analysis and the two-step clustering have been performed with SPSS©.

4. RESULTS

Our dataset consists of ninety seven (97) java open source games of various size, defect frequency, bug fixing efficiency and design pattern use intensity. The descriptive measurements that outline our dataset are presented in Table 1 (minimum, maximum, mean value and standard deviation).

Table 1. Descriptive Statistics of Dataset

Variable	min	max	mean	std. dev
bugs opened	10	253	59,82	48,20
bugs fixed	3	228	59,76	44,49
bugs fixed / bugs opened	0.15	3.26	1.10	0.572
number of classes	37	1964	909,38	464,42
overall pattern participants	12.46%	50.51%	30.74%	6.99
factory instances	0	24	4,95	4,73
factory participants	0	136	19,44	28,14
factory participants in %	0,00%	6,95%	1,53%	1,63%
singleton instances	0	63	23,77	15,609
singleton participants	0	63	23,77	15,609
singleton participants in %	0,00%	21,62%	3,04%	3,27%
composite instances	0	7	1,16	1,320
composite participants	0	66	5,14	10,831
composite percentage in %	0,00%	3,36%	0,45%	0,69%
adapter instances	1	224	98,58	52,409
adapter participants	2	264	105,24	56,638
adapter percentage in %	3,77%	24,74%	11,95%	4,76%
observer instances	0	15	7,90	5,011
observer participants	0	152	52,78	53,828
observer percentage in %	0,00%	21,82%	5,22%	4,76%
State/Strategy instances	1	242	109,98	79,206
State/Strategy participants	2	550	163,86	114,709
State/Strategy percentage in %	4,12%	34,02%	16,70%	6,03%
template instances	0	27	11,68	5,878
template participants	0	155	75,82	53,869
template percentage in %	0,00%	16,37%	7,76%	4,84%
decorator instances	0	26	2,62	4,338
decorator participants	0	105	13,49	17,816
decorator percentage in %	0,00%	5,36%	1,28%	1,07%
prototype instances	0	411	14,07	62,375
prototype participants	0	464	28,54	79,063
prototype percentage in %	0,00%	24,33%	2,02%	4,22%
proxy instances	0	23	10,66	8,807
proxy participants	0	37	12,53	9,931
proxy percentage in %	0,00%	9,09%	1,48%	1,52%
Proxy2 instances	0	2	0,31	,727
Proxy2 participants	0	4	0,62	1,454
Proxy2 percentage in %	0,00%	0,28	0,04%	0,10%

Additionally, we performed Pearson x^2 tests, so as to investigate the correlation between dependent variables (i.e. defect frequency and debugging efficiency) and the numerical independent variables (i.e. pattern instances, pattern participants in classes, and percentage of project classes that participate in every instance of the pattern). The statistically significant results are presented in Table 2.

In order to visualize the impact of each independent variable on the dependent variables we created boxplots on the most important correlations. The boxplots are presented in Figures 1 – 11. In a boxplot the bold line inside the box represent the mean value of the dependent variable (y-axis) in the corresponding value of the grouping-indepentent variable (x-axis). Additionally, the box covers 50% of the cases. The rest 50% of the cases are divided into two equal groups (25% each) that are represented by the lines starting from the top and the bottom of the box. Finally, outliers, when they exist, are represented by circles outside the boxes.

Table 2. Descriptive Statistics of Dataset

Variable	test	bugs opened	debugging efficiency
Factory instances	pearson correlation	-,367**	,011
	sig. (2-tailed)	,000	,918
Factory participants	pearson correlation	-,270**	-,037
	sig. (2-tailed)	,008	,721
Factory percentage	pearson correlation	-,293**	,039
	sig. (2-tailed)	,004	,705
Singleton participation pct	pearson correlation	-,052	-,217*
	sig. (2-tailed)	,612	,033
Composite instances	pearson correlation	-,398**	,045
	sig. (2-tailed)	,000	,663
Composite participants	pearson correlation	-,234*	-,106
	sig. (2-tailed)	,021	,304
Composite participation pct	pearson correlation	-,203*	-,059
	sig. (2-tailed)	,046	,565
Adapter participants	pearson correlation	,276**	-,109
	sig. (2-tailed)	,006	,289
Adapter participation pct	pearson correlation	,531**	-,224*
	sig. (2-tailed)	,000	,028
Observer instances	pearson correlation	-,397**	,185
	sig. (2-tailed)	,000	,070
Observer participants	pearson correlation	-,259*	,163
	sig. (2-tailed)	,010	,111
Observer participation pct	pearson correlation	-,338**	,219*
	sig. (2-tailed)	,001	,032
State/ Strategy instances	pearson correlation	-,335**	,173
	sig. (2-tailed)	,001	,091
Template participants	pearson correlation	,253*	,014
	sig. (2-tailed)	,012	,891
Template participation pct	pearson correlation	,511**	-,014
	sig. (2-tailed)	,000	,892
Decorator participation pct	pearson correlation	,191	-,212*
	sig. (2-tailed)	,060	,037
Prototype participants	pearson correlation	-,205*	-,103
	sig. (2-tailed)	,044	,315
Prototype participation pct	pearson correlation	-,242*	-,038
	sig. (2-tailed)	,017	,710
Proxy instances	pearson correlation	-,448**	,193
	sig. (2-tailed)	,000	,058
Proxy participants	pearson correlation	-,427**	,140
	sig. (2-tailed)	,000	,173
Proxy participation pct	pearson correlation	-,365**	,132
	sig. (2-tailed)	,000	,198

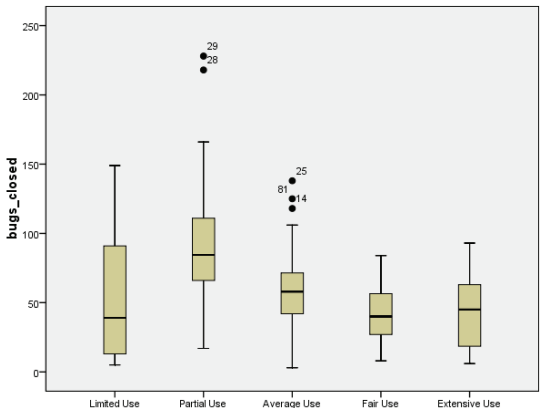

Figure 1. Boxplot on Defect Frequency and Factory Pattern Usage

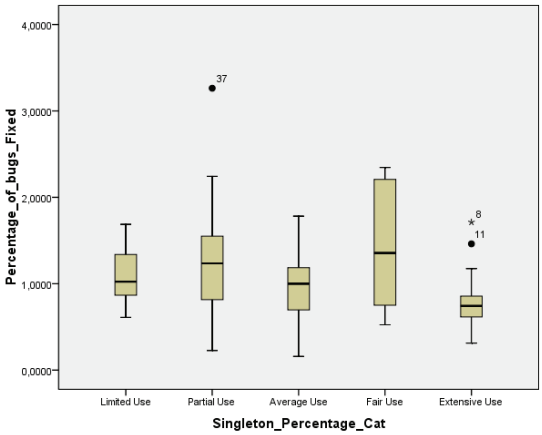

Figure 2. Boxplot on Debugging and Singleton Pattern Usage

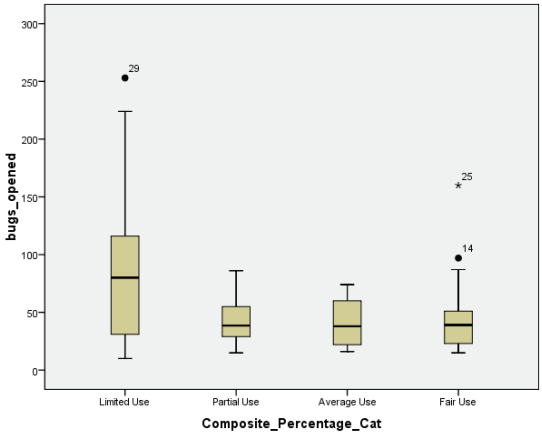

Figure 3. Boxplot on Defect Frequency and Composite Pattern Usage

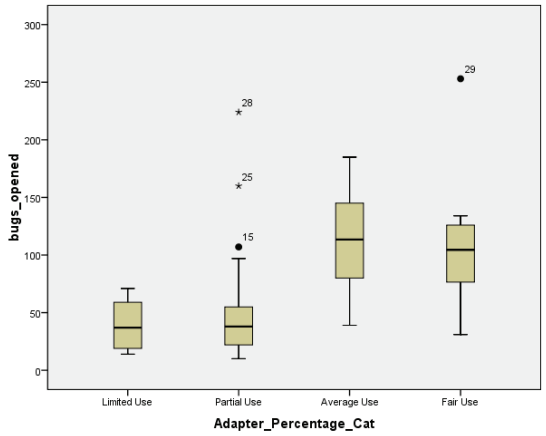

Figure 4. Boxplot on Defect Frequency and Adapter Pattern Usage

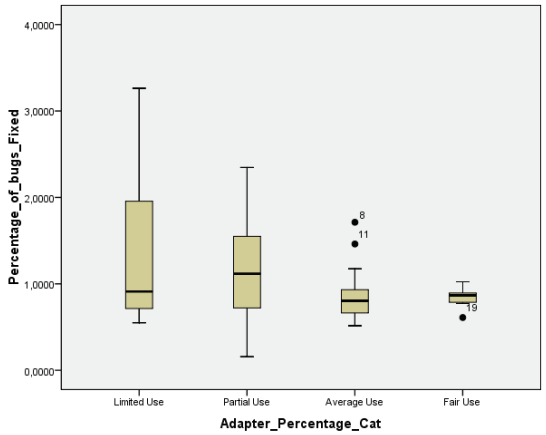

Figure 5. Boxplot on Debugging and Adapter Pattern Usage

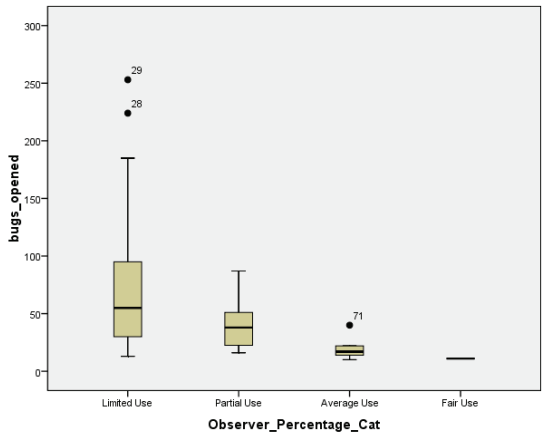

Figure 6. Boxplot on Defect Frequency and Observer Pattern Usage

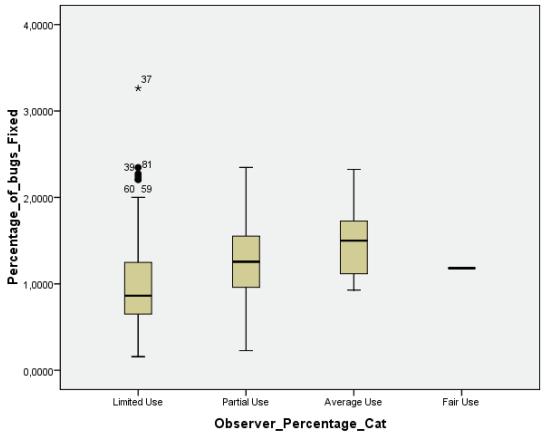

Figure 7. Boxplot on Debugging and Observer Pattern Usage

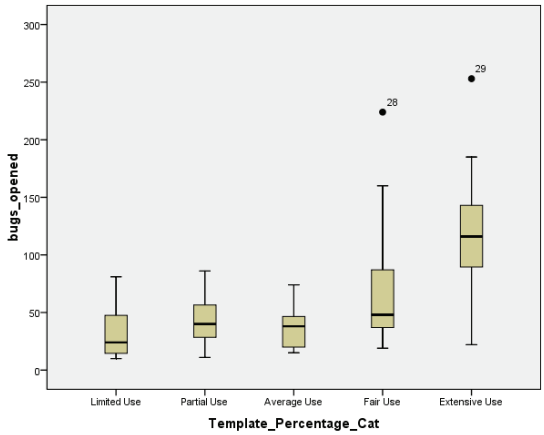

Figure 8. Boxplot on Defect Frequency and Template Pattern Usage

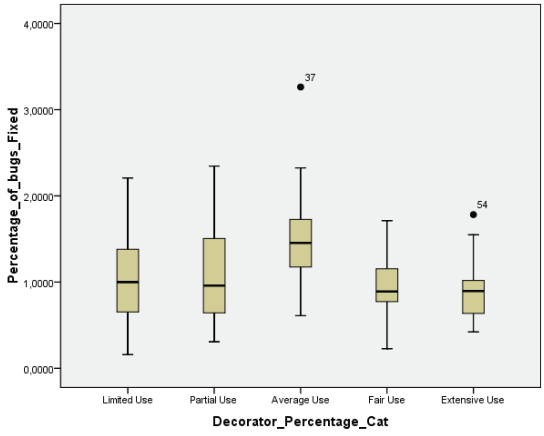

Figure 9. Boxplot on Debugging and Decorator Pattern Usage

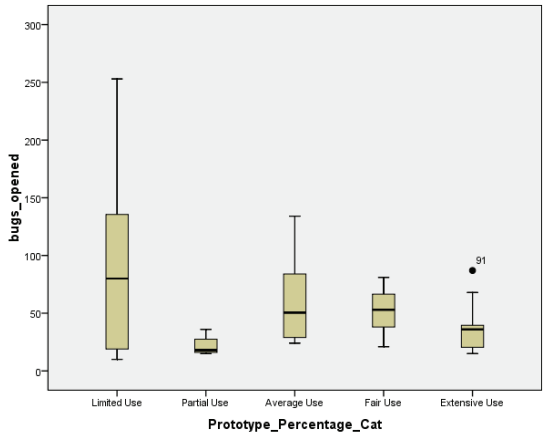

Figure 10. Boxplot on Defect Frequency and Prototype Pattern Usage

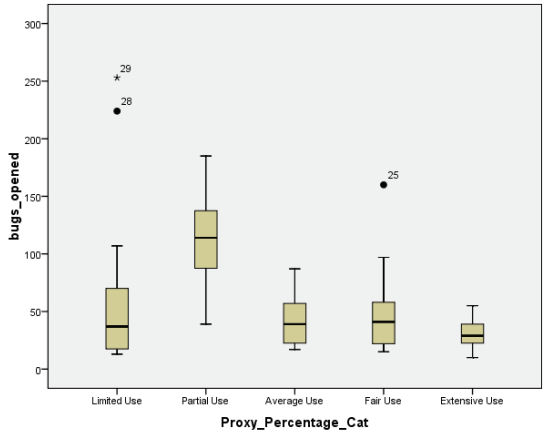

Figure 11. Boxplot on Defect Frequency and Proxy Pattern Usage

Finally, in order to explore if there is a statistically significant difference in the debugging efficiency among different pattern application intensity we performed several independent sample t-tests. The most important findings are reported in Table 3.

Table 3. Difference in Mean Values of Debugging Efficiency among Patterns and their Application Intensity

Pattern	Application Intensity	diff	sig
Factory	Partial Use – Average Use	-0.441	0.02
	Partial Use – Fair Use	-0.366	0.05
Singleton	Limited Use – Fair Use	-0.374	0.04
	Limited Use – Extensive Use	0.294	0.00
	Average Use – Fair Use	-0.484	0.01
	Average Use – Extensive Use	0.184	0.00
Adapter	Partial Use – Extensive Use	0.428	0.00
	Fair Use – Extensive Use	0.403	0.01
Observer	Limited Use – Extensive Use	-0.560	0.00
	Average Use – Extensive Use	-0.444	0.00
	Fair Use – Extensive Use	-0.412	0.05
Template Method	Partial Use – Fair Use	-0.405	0.05
	Fair Use – Extensive Use	0.526	0.00
Decorator	Average Use – Fair Use	0.554	0.00
	Average Use – Extensive Use	0.595	0.00
Prototype	Limited Use – Average Use	-0.365	0.01
Proxy	Partial Use – Average Use	-0.262	0.03
	Partial Use – Fair Use	-0.121	0.01

5. DISCUSSION

This section of the paper discusses the findings of our study concerning the research questions stated in section 3.1.

5.1 Design Patterns and Defect Frequency

The empirical results of our study indicate that design pattern application frequency is related to a significant extent to the defect frequency of computer games. The fact that the overall number of pattern instances and the overall number of classes that participate in patterns is not significantly correlated to the number of defects ($sig_{instances}$ = 0.06 and $sig_{participants}$ = 0.35), suggests that there are certain characteristics of specific design patterns, which influence defect frequency.

More specifically, Abstract Factory, Singleton, Composite, Observer, State, Strategy, Prototype and Proxy have been indicated as patterns that are negatively correlated to defect frequency. Thus, it appears that as the number of instances of such patterns increases the number of open bugs in a project decreases. On the other hand, Adapter and Template Method patterns are positively correlated to defect frequency, i.e. as the number of Adapter and Template increase the number of bugs appears to increase as well. Although these results are statistically significant they need further investigation, especially the results on Adapter and Template Method, because they might be influenced from the confounding factors of the study.

A possible explanation for the results on Composite, Observer and Prototype patterns is that, since they are quite complex in their structure, there is a probability to be applied by more experienced developers, leading to less errors. On the other hand, the simplicity of Factory, Singleton, Proxy, State and Strategy seems to help developers avoid implementation defects.

On the contrary, the extended use of the Adapter pattern might insert defects in the system, because of the code that is being reused. Firstly, often the developers of the system are not familiar with the piece of code that they are adapting and secondly, the defects of the adapted code are added to the defects of the target system. Similarly, Template Method might produce errors that derive from the pattern's structure, i.e. the deep inheritance tree it involves.

5.2 Design Patterns and Bug Fixing

Similarly to the correlation of total number of open bugs, the bug fixing rate is not correlated to the total number of pattern instances and participants ($sig_{instances}$ = 0.86 and $sig_{participants}$ = 0.15).

However, there are some patterns that appear to have an effect on bug fixing rate. More specifically, Singleton, Adapter, Observer and Decorator appear to be statistically significantly correlated to debugging rate.

Singleton pattern appear to be negatively correlated to debugging efficiency, i.e. as the number of singleton instances increase, the rate of bug fixing decreases. The use of a singleton pattern is

similar to global variables, in the sense that the instance returned by the singleton pattern is accessible from many objects and changes to singleton are expected to have large impact on the system. Therefore, debugging a system that involves many singleton pattern instances is expected to be more complicated and hence difficult to manage.

Additionally, the extensive use of adapter pattern instances appears to hamper bug fixing activities. The most obvious explanation of this phenomenon is that the developers are not fully aware of the code that they are reusing and therefore bug fixing on such code fragments is harder. Moreover, the Decorator pattern was found to be negatively correlated to debugging efficiency. This result may occur because the call graph of a decorator pattern structure is difficult to understand and maintain.

On the other hand, using the observer pattern enhances the debugging procedure, since it provides a well structured way of managing with interaction among application layers, such as user interfaces, game controls and game logic. Such interactions are expected to be quite complex without the existence of the pattern, since the discrimination of layers increase system's modularity. Therefore, the higher the number of classes that participate in the observer is, the higher the debugging efficiency.

6. THREATS TO VALIDITY

This section deals with presenting the threats to the validity of our work. In any empirical study there are several threats to validity if one attempts to generalize the results outside the scope of the study. In our study, the results cannot be generalized to all 23 GoF patterns, but only to the 11 that we have examined. Additionally, the results cannot be straightforwardly valid for closed source software, for games written in programming languages other than java and for open-source domains, other than games.

7. CONCLUSIONS – FUTURE WORK

This work aimed at identifying possible correlations between the application rate of design patterns, the defect frequency and the debugging efficiency in open-source games. For this reason we conducted a case study on 97 open-source java games. The results of our study suggested that several design patterns are correlated to the number of bugs that are reported in java open source games, while others are correlated to the rate of bug fixing activities.

For instance, the Adapter pattern is indicated as a pattern that has negative effect on both defect frequency and debugging efficiency. A possible intuitive explanation of this result is that adapter is most commonly used in code reuse activities. More specifically, when reusing code a developer might not have a full understanding of the code that he is reusing. Thus, he has problems in fixing bugs in a piece of code that he is not fully aware of. Simultaneously, possible bugs of reused parts are added in open bugs of the target system. On the other hand, an increased amount of Observer pattern instances lead to a decrease in the bugs that are open in an open source game and accelerate the debugging procedure. A possible explanation on this is that developers who are using the Observer pattern are experienced on design activities, and therefore less error prone. Additionally, observer clearly demarcates the modules of the game and therefore debugging activities are enhanced.

It is not clear however, whether the use of design patterns is the root-cause of the defect detection and removal performance of java open source game projects. It might be the case that the use of design patterns is related to other critical project characteristics, such as community level of experience, project vivacity etc. Such issue is subject of further investigation.

Future research plans include the replication of the case study on a greater variety of projects, across different domains. Such an attempt will provide deeper understanding on whether the results of this study are game related or not. Additionally, the type of defects is going to be assessed and their severity is going to be correlated to patterns as well.

8. REFERENCES

[1] A. Ampatzoglou and A. Chatzigeorgiou, "Evaluation of object-oriented design patterns in game development", *Information and Software Technology*, Elsevier, 49 (5), pp. 445-454, May 2007

[2] A. Ampatzoglou, I. Stamelos, "Software engineering research for computer games: A systematic review" *Information and Software Technology*, 52 (9): 888-901 (2010)

[3] V.R. Basili, R.W. Selby, D.H. Hutchens, 1986, *In IEEE Transactions on Software Engineering,* "Experimentation in Software Engineering", IEEE Computer Society

[4] B. Baudry, Y. Le Traon, G. Sunye and J. M. Jezequel, "Measuring and Improving Design Patterns Testability", *Proceedings of the 9th International Symposium on Software Metrics ,* IEEE, pp. 50, Sydney, Australia, 03-05 September 2003

[5] S. Bjork and J. Holopainen, "Patterns in game design", Game Development Series, *Charles River Media*, 2004

[6] E. Gamma, R. Helms, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, *Addison-Wesley Professional*, Reading, MA, 1995

[7] M. Gatrell, S. Counsell and T. Hall, "Design Patterns and Change Proneness: A Replication Using Proprietary C# Software", *Proceedings of the 2009 16th Working Conference on Reverse Engineering,* pp. 160-164, Lille, France, 13-16 October 2009

[8] M. Elish, "Do Structural Design Patterns Promote Design Stability?", *Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 01 (COMPSAC'06),* IEEE, pp 215-220, Chicago, Illinois, 17-21 September 2006

[9] B. Ellis, J. Stylos and B. Myers, "The Factory Pattern in API Design: A Usability Evaluation", *Proceedings of the 29th international conference on Software Engineering,* IEEE, pp. 302-312, Minneapolis, Minnesota, 20-26 May 2007

[10] S. Jeanmart, Y.G. Gueheneuc, H. Sahraoui and N. Habra, "A Study of the Impact of the Visitor Design Pattern on Program Comprehension and Maintenance Tasks", *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09),* IEEE, p.p. 69-78, Lake Buena Vista, Florida, 15-16 October 2009.

[11] F. Khomh and Y.G. Gueheneuc, "Do design patterns impact software quality positively?", *IEEE Proceedings of the 12th European Conference on Software*

Maintenance and Reengineering (CSMR 2008), pp.274-278, 1-4 April 2008, Athens, Greece

[12] B. Kitchenham, L. Pickard, S.L. Pfleeger, 1995, *In IEEE Software*, "Case Studies for Method and Tool Evaluation".

[13] K. Kouskouras, A. Chatzigeorgiou and G. Stephanides, "Facilitating software extension with design patterns and Aspect-Oriented Programming", *Journal of Systems and Software,* Elsevier, 81 (10), pp 1725-1737, October 2008.

[14] A. McWilliams, T. Reicher, G. Klinker and B. Bruegge, "Design Patterns for Augmented Reality Systems", *Proceedings of the 2004 International Workshop Exploring the Design and Engineering of Mixed Reality Systems (MIXER' 04)*, pp. 1-8, Funchal, Madeira, 13 January 2004.

[15] D. Z. Nguyen, S. B. Wong, "Design Patterns for Games", *Special Interest Group on Computer Science Education (SIGCSE'02)*, Association of Computing Machinery, pp. 126- 130, Cincinnati, Kentucky, 27 February – 2 March 2002.

[16] L. Prechelt, B. Unger-Lamprecht, W .F. Tichy, P. Brossler and L. G. Votta, "A controlled experiment in maintenance comparing design patterns to simpler solutions", *IEEE Transactions on Software Engineering,* IEEE, 27 (3), pp 1134 -1144 , December 2001

[17] H. Rajan, S. M. kautz and W. Rowcliffe, "Concurrency by Modularity: Design Patterns, a Case in Point", *Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '10), ACM,* p.p. 790-805, Reno, Nevada, 17-21 October 2010.

[18] T.M. Rhyne, P. Doenges, B. Hibbard, H. Pfister, N. Robins, "The impact of Computer Games on scientific & information visualization: "if you can't beat them, join them" (panel)", *IEEE Visualization, Proceedings of the conference on Visualization '00*, Salt Lake City, Utah, USA, pages 519-521

[19] A. Rollings, D. Morris, "Game Architecture and Design: A New Edition", *New Riders*, Indianapolis, 2003.

[20] R. Rucker, "Software engineering and computer games", *Addison Wesley*, Essex, United Kingdom, 2003

[21] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides and S.T. Halkidis, 2006. *In IEEE Transaction on Software Engineering,* "Design Pattern Detection using Similarity Scoring", IEEE Computer Society

[22] L. Valente, A. Conci, "Guff: A Game Development Tool", *Digital version of the proceedings of XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI),* Natal, Brazil, 2005

[23] M. Vokáč, W. Tichy, D. I. K. Sjøberg , E. Arisholm and M. Aldrin, "A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns - A Replication in a Real Programming Environment", *Empirical Software Engineering, Springer,* 9(3), pp 149-195, September 2004

[24] M. Vokáč, "Defect Frequency and Design Patterns: An Empirical Study of Industrial Code", *IEEE Transactions on Software Engineering*, IEEE, 30(12), pp. 904-917, December 2004

[25] P. Wendorff, "Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project", *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, IEEE, pp. 77, Lisbon, Portugal , 14-16 March 2001

[26] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, A. Wesslen, "Experimentation in Software Engineering", *Kluwer Academic Publishers*, Boston/Dordrecht/ London, 1st edition, 2000