AN EMPIRICAL STUDY ON DESIGN PATTERN USAGE ON OPEN-SOURCE SOFTWARE

Apostolos Ampatzoglou, Kyriaki Savva, Ioannis Stamelos
Aristotle University of Thessaloniki, Thessaloniki, Greece
e-mails: apamp@csd.auth.gr, stamelos@csd.auth.gr

Sofia Charalampidou
Technological Educational Institute of Thessaloniki, Greece
email: sofchar@it.teithe.gr

Keywords: open source software; design patterns; empirical study;

Abstract Currently, open source software communities are thriving and the number of projects that are available through well known code repositories is rapidly increasing over the years. The amount of code that is freely available to developers facilitates high reuse opportunities. One of the major concerns of developers when reusing code is the quality of the code that is going to be reused. Design patterns are well known solutions that are reported to produce substantial benefits with respect to software quality. In this paper, we investigate the extent to which design patterns are employed in open source software. More specifically, this study reports empirical results based on the number and type of design patterns retrieved from open source software projects. Up to now, one hundred and eight (108) open source software projects of various characteristics have been considered. The results of the study suggest that several patterns are more frequently used in open source software than others, that some patterns are more applicable in some categories than others and that program size, number of downloads, days of project activity and the number of developers are crucial factors that influence the use of design patterns in open source software project.

1. INTRODUCTION

Open source software (OSS) development process, which was introduced in 1998 (Feller and Fitzgerald, 2002), is a quite modern trend in software production. Despite its short life period the open source software community, can exhibit some extremely successful project with great acceptance in computer communities, such as Linux, Apache Server and Mozilla Firefox.

The development of an open source project is based on collaboration. A single developer, or a group of developers, starts a project and releases a version that is freely available, over the internet, for use and modification. Then, the open source community extends and maintains the project. This type of development has both advantages and disadvantages. One disadvantage of open source software development is the lack of documentation and technical support. Whereas, the main advantages of open source software is their low cost, their reliability and the fact that they provide their source code to the user, in order for him to be able to customize the software according to his special needs (Samoladas, Stamelos, Angelis and Oikonomou, 2004).

Furthermore, open source software provides great reuse opportunities, in the sense that a wide variety of code is freely available for developers. In order for a code segment to be easily and successfully adopted by another project it should be understandable, easily maintainable and flexible. Design patterns have been introduced in 1995 in (Gamma, Helms, Johnson and Vlissides, 1995) as common solutions to common design problems. The main motivation when introducing patterns was to provide a common vocabulary to developers, which match to reusable, flexible and maintainable design solutions. In addition to that, in (Meyer and Arnout, 2006, Arnout and Meyer, 2006), the authors describe how object-oriented design patterns can be transformed to reusable components.

In the literature, a great variety of studies have attempted to assess the impact of design pattern application on software quality. These studies, mostly empirical ones, suggest that object oriented

design patterns are not universally good or bad. A more detailed presentation of the current state of the art discussing the effect of design pattern application on software quality is presented in section 2.

This paper aims at investigating the use of object-oriented design patterns in open-source software. More specifically we have employed an empirical methodology, i.e. a case study, so as to assess which patterns are more frequently applied in open-source software, which differences appear within software domains and which are the most influential factors on pattern application.

In the next section of the paper, we provide a literature review on design patterns effect on software quality. In section 3, the methodology of our work, i.e. research questions, case study process and data analysis methods, is presented. In section 4, we present the findings of our empirical study. Additionally, section 5 provides a discussion on the results, grouped according to the research question they refer to. Finally, by the end of the paper we present possible threats to validity, future work and conclusions.

2. DESIGN PATTERNS

This section of the paper presents the findings of a literature review on the effect of design pattern application on software quality. Software quality is commonly divided into internal and external quality (Bansiya and Davis, 2002). Software internal quality is measurable and assesses software characteristics such as complexity, cohesion, coupling, inheritance etc, which are not easily understandable from the end-user or the developer. On the other hand, external quality is not directly measured but it is closer to the user's and the developer's sense. The most well known external quality attributes, i.e. functionality, reliability, usability, efficiency, maintainability and portability, are described in ISO/IEC 9126.

The effect of design patterns on software internal quality was investigated in (Ampatzoglou and Chatzigeorgiou, 2007, Huston, 2001). According to (Huston, 2001), coupling is reduced by using the Mediator pattern, the DIT and NOC metrics are reduced by using the Bridge pattern and finally the project's complexity with respect to number of methods is reduced by using the Visitor pattern. The results of (Ampatzoglou and Chatzigeorgiou, 2007) suggest that coupling and complexity, with respect to cyclomatic complexity, are reduced and cohesion among methods is increased by applying the State and the Bridge pattern. As a side-effect the project size concerning the number of classes increases.

Furthermore, several studies have investigated the effect of design patterns in external quality. Vokac et.al and Prechelt et.al (Prechelt, Unger, Tichy, Brossler and Votta, 2001, Vokác, Tichy, Sjøberg, Arisholm and Aldrin, 2003) investigated the effect of design patterns i.e. Abstract Factory, Observer, Decorator, Composite and Visitor to software maintainability by conducting controlled experiments. According to the experiment, the application of a design pattern is usually more useful than the simpler solution. The software engineer should select between employing a design pattern or a simple solution according to common sense. Furthermore, in (Hsueh, Chu and Chu, 2008), the writers discuss how design patterns affect one quality attribute, which is the most obvious attribute that the pattern has effect on. The quality attribute is selected according to the pattern's non functional requirements, whereas the metric is selected according to (Bansiya and Davis, 2002).

An industrial case study is presented in (Wendorff, 2001), where severe maintainability problems have been caused by inappropriate pattern application. The author classifies the reasons of inappropriately using design patterns into two categories (1) software developers have not realised the rationale behind the patterns that they have applied and (2) the patterns that have been used, have not satisfied the project's requirements. Additionally, the paper underlines the need for documenting pattern usage and that pattern removal results in extreme cost. In (Khomh and Gueheneuc, 2008), the authors performed an analysis on software maintenance, with professional engineers. The empirical study concluded that design patterns do not always have positive effect on software quality. More specifically, it is suggested that when patterns are used, the simplicity, the learnability and the understandability are negatively influenced.

In (Bieman, Jain and Yang, 2001), the writers conducted an industrial case study, in order to investigate the correlation among code changes, reusability, design patterns, and class size. The results of the study suggest that the number of changes is highly correlated to class size and that classes that play roles in design patterns or that are reused through inheritance are more change prone than others. Although this study is well-structured and validated, it refers to a specific maintainability issue, change proneness and does not examine maintainability aspects such as change effort and design quality. In (Di Penta, Cerulo, Gueheneuc and

Antoniol, 2008) the authors have aimed to investigate correlations among class change proneness, the role that a class holds in a pattern and the kind of change that occurs. Three open source projects were used in order to conduct the empirical study. The results of the paper comply with common sense in the majority of design patterns. However, there are cases where the obtained results differ from the expected ones.

3. METHODOLOGY

According to (Wohlin, Runeson, Host, Ohlsson, Regnell and Wesslen, 2000), there are three major empirical investigation approaches, surveys, case studies and experiments. Considering the nature and the subject of our research we have selected to conduct a case study. The plethora of open source projects consist case studies as the optimum research approach. On the contrary, surveys are not suitable for our research because in this case we would miss the patterns that were employed without intension by programmers. Finally, an experiment with open-source programmers would decrease the number of subjects in our research.

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 (Kitchenham, Pickard and Pfleeger, 1995). According to (Kitchenham, Pickard and Pfleeger, 1995) 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 (Kitchenham, Pickard and Pfleeger, 1995) 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 from our study.

RQ1: Which design pattern is more frequently applied in open source software?

RQ2: Are there any differences in pattern application within the software categories under study?

RQ3: Which are the most influential factors in pattern application?

3.2 Case Study Plan

According to [Basili, Selby and Hutchens, 1986], in order to produce a solid methodology for an empirical validation method, a study plan should be thoroughly prepared. In this case study the plan involved a five step procedure:

1. choose open source project categories
2. identify a number of projects that fulfil certain selection criteria, for each category
3. perform pattern detection for every selected project
4. tabulate data
5. analyze data with respect to the research questions

In this study the OSS project categories that have been considered are object-oriented development tools, e-commerce applications and computer games. The three categories have been selected for different reasons. Firstly, the object-oriented development tools category has been selected as a highly active topic in open source communities (Sowe, Angelis, Stamelos and Manolopoulos, 2007) and because the developers of this category are expected to be aware of design patterns, due to the nature of the software they develop. Secondly, the computer games category has been employed because it is also an active topic (Sowe, Angelis, Stamelos and Manolopoulos, 2007) and on contrary to the first category its developers are more probable to write code without prior design activities (McShafrrey, 2003). Finally, the e-commerce applications have been selected as a topic with smaller activity in OSS development communities.

From these categories we have selected projects that fulfilled the following criteria:

1. software written in java, due to limitations of pattern detection tool (Tsantalis, Chatzigeorgiou, Stephanides and Halkidis, 2006)
2. software that provides binary code, due to limitations of pattern detection tool
3. software that has been downloaded more than 100 times, in order to be considered active
4. software that has more than 20 classes, in order not to be considered trivial

The complete list of projects that are used in the case study is presented in the Appendix by the end of the paper. The number of software considered in each category was not the equal, due to the limited number of projects that fulfilled the inclusion criteria in the e-commerce category.

In case studies, factors, different than the independent variables, which influence the value of the dependent variable, are regarded as confounding factors. Some possible confounding factors that are expected to affect design pattern application are the programming experience of the developers and the educational status of the developers concerning software engineering issues. Such data could not have been retrieved in a case study, where the data on the subjects are gathered through observation, but only through a controlled experiment (Wohlin, Runeson, Host, Ohlsson, Regnell and Wesslen, 2000). On the other hand, it is expected that in a random developer sample of a large developers' community, the distribution of skill and experience are closely near to the distribution of the population.

3.3 Data Analysis Methods

The dataset that has been created after design pattern detection involved only numerical data. But, some techniques that were employed during data analysis need categorical or binary variables. Thus, certain data transformations have taken place, as presented in Appendix B. On the completion of the pre-processing phase each project was characterized by 57 variables:

1. s/n
2. name
3. category
4. latest release year
5. days active
6. number of downloads
7. number of developers
8. number of versions
9. number of classes
10. number of factory method instances
11. number of prototype instances
12. number of singleton instances
13. number of creational pattern instances
14. number of adapter instances
15. number of composite instances
16. number of decorator instances
17. number of proxy instances
18. number of structural pattern instances
19. number of observer instances
20. number of state-strategy instances
21. number of template method instances
22. number of visitor instances
23. number of behavioural pattern instances
24. 20 categorical variables for variables 4-23
25. 14 binary variables for variables 10-23

The analysis phase of our study has employed techniques from statistics and clustering. These techniques are:

- descriptive statistics
- independent sample t-test
- paired sample t-test
- K-Means clustering
- Pearson x^2 test

Concerning *RQ1*, we have employed descriptive statistics and paired sample t-tests so as to compare the mean number of instances for each design pattern. In the investigation of *RQ2*, for similar reasons we have used descriptive statistics and independent sample t-tests. Finally, concerning *RQ3*, we have performed Pearson x^2 test and K-Means clustering. The statistical analysis and the two-step clustering have been performed with SPSS©.

4. RESULTS

In Table 1, the mean number of design pattern instances is presented. The data concern the whole data set without discrimination across software categories. In addition to that, the maximum number of patterns that appear in one project is also provided. Finally, the standard deviation of each variable is presented.

Table 1: Average Number of Pattern Instances

	Maximum	**Mean**	**Std. Deviation**
Factory	6	0.55	1.08
Prototype	12	0.45	1.63
Singleton	55	4.83	9.35
Creational	*56*	*5.83*	*10.15*
Adapter	223	11.39	31.42
Composite	2	0.09	0.32
Decorator	7	0.49	1.32
Proxy	11	0.58	1.75
Structural	*232*	*12.56*	*33.09*
Observer	4	0.40	0.92
State	149	9.42	22.18
Template	10	1.54	2.17
Visitor	4	0.05	0.40
Behavioural	*151*	*11.40*	*23.45*

The results of Table 1 provide indications on the applicability of each pattern in OSS. In order to be

able to compare the mean values of each variable in a more elaborate way, we have performed 55 paired sample t-tests, i.e. one test for every possible pair of design patterns. The results of a t-test between two variables are interpreted by two numbers, the mean difference (diff) and the t-test significance (sig). The *diff* variable represents the difference of subtracting the mean value of the second variable, from the mean value of the first. Whereas, *sig* represents the possibility, that *diff* is not statistically significant. In Table 2, we present the statistically significant differences in pattern application.

Table 2: Significant paired sample t-tests on pattern employment difference

	diff	**Sig**
Factory – Singleton	-4,29	0.00
Factory – Adapter	-10,84	0.00
Factory – Composite	0,45	0.00
Factory – State	-8,87	0.00
Factory – Visitor	0,50	0.00
Factory – Template	-0,99	0.00
Prototype – Singleton	-4,38	0.00
Prototype – Adapter	-10,94	0.00
Prototype – Composite	0,36	0.02
Prototype – State	-8,96	0.00
Prototype – Template	-1,08	0.00
Prototype – Visitor	0,41	0.01
Singleton – Adapter	-6,56	0.01
Singleton – Composite	4,74	0.00
Singleton – Decorator	4,34	0.00
Singleton – Proxy	4,25	0.00
Singleton – Observer	4,44	0.00
Singleton – State	-4,58	0.01
Singleton – Template	3,30	0.00
Singleton – Visitor	4,79	0.00
Adapter – Composite	11,30	0.00
Adapter – Decorator	10,90	0.00
Adapter – Proxy	10,81	0.00
Adapter – Observer	10,99	0.00
Adapter – Template	9,85	0.00
Adapter – Visitor	11,34	0.00
Composite – Decorator	-0,40	0.00
Composite – Observer	-0,31	0.00
Composite – Proxy	-0,49	0.00
Composite – State	-9,32	0.00
Composite – Template	-1,44	0.00
Decorator – State	-8,93	0.00
Decorator – Template	-1,05	0.00
Decorator – Visitor	0,44	0.00
Proxy – State	-8,83	0.00
Proxy – Template	-0,95	0.00
Proxy – Visitor	0,54	0.00
Observer – State	-9,02	0.00
Observer – Template	-1,14	0.00
Observer – Visitor	0,35	0.00
State – Template	7,88	0.00
State – Visitor	9,37	0.00
Template – Visitor	1,49	0.00

In Table 3, the mean numbers of instances of each design patterns within the software categories under study are presented. In this table, the maximum values and the standard deviation variables have been omitted due to space limitations.

Table 3: Average Number of Pattern Instances among Software Categories

	OO Dev Tools	**E – Commerce**	**Games**
Factory	0.74	0.50	0.40
Prototype	0.52	0.43	0.40
Singleton	4.05	4.14	5.65
Creational	*5.31*	*5.07*	*6.46*
Adapter	5.10	10.00	16.85
Composite	0.19	0.00	0.04
Decorator	0.55	0.57	0.42
Proxy	0.50	0.00	0.81
Structural	*6.33*	*10.57*	*18.12*
Observer	0.36	0.07	0.52
State	7.48	13.50	9.88
Template	1.95	1.00	1.35
Visitor	0.12	0.00	0.00
Behavioural	*9.90*	*14.57*	*11.75*

In order to statistically validate the results of the above table, we performed 42 independent sample t-tests, i.e. one test for each pattern for all the possible pairs of software categories. In Table 4, we provide the statistically significant results on comparing pattern application between software categories. The results are presented similarly to those of Table 2.

Table 4: Significant independent sample t-tests

	Pattern	**diff**	**Sig**
OO Dev - Ecommerce	Composite	0,19	0.01
OO Dev - Ecommerce	Proxy	0,50	0.02
OO Dev – Ecommerce	Observer	0,29	0.05

OO Dev - Ecommerce	Template	0,95	0.04
OO Dev – Games	Adapter	-11,75	0.05
OO Dev – Games	Composite	0,15	0.05
Games – Ecommerce	Proxy	0,81	0.01
Games – Ecommerce	Observer	0,45	0.01

In Table 5, we present some general statistics of the software categories under study. The numerical variables are represented by their mean value, whereas the categorical variables are represented by their mode value. After validating the results through t-tests the only statistical significant differences have proven to be the days of activity between games and e-commerce, the days of activity between OO development tools and e-commerce and the number of versions between OO development tools and e-commerce.

Table 5: General Statistics on Software Categories

	OO Development	**E – Commerce**	**Games**
Release Year	2009	2008/2009	2009
Days Active	524.29	107.00	801.63
Downloads	10945.48	1830.93	28694.25
Developers	1.29	1.43	2.92
Versions	5.93	3.00	5.63
Classes	108.07	243.79	171.65

Table 6 presents the significance values of the correlations between the number of pattern instances in the three pattern categories, i.e. creational, behavioural and structural, and all the non-pattern related variables, i.e. software category, release year, days of activity, downloads, number of developers, versions and size. The test that was employed for investigating the correlation between each set of variables is the Pearson chi-square test.

Table 6: Person x^2-test Correlations

	Creational	**Behavioural**	**Structural**
Category	0.58	0.72	0.09
Release Year	0.92	0.85	0.57
Days Active	0.00	0.00	0.00
Downloads	0.00	0.01	0.00
Developers	0.00	0.00	0.00
Versions	0.06	0.03	0.01
Classes	0.00	0.00	0.00

In order to investigate and create profiles on pattern usage, we have created clusters with the use of the K-Means algorithm. The five clusters that the algorithm has created are presented in Table 7. The data of the table correspond to the categorical variables that are referenced in Appendix B.

Table 7: Software Profiles on Pattern Usage with Categorical Variables of Appendix B

Cluster No	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**
Release year	2	2	2	3	3
Days Active	4	3	1	3	2
Downloads	3	2	2	3	2
Developers	1	1	0	0	0
Versions	2	2	2	2	1
Classes	4	4	3	2	1
Factory	2	2	1	1	1
Prototype	1	1	1	1	1
Singleton	5	3	2	2	1
Adapter	5	5	3	2	1
Composite	1	1	1	1	1
Decorator	2	1	1	1	1
Proxy	2	2	1	1	1
Observer	2	1	1	1	1
State	5	5	3	2	1
Template	3	2	2	2	1
Visitor	1	1	1	1	1
Category	2	0	1	1	1

5. DISCUSSION

This section of the paper discusses the results of the statistical and clustering techniques that have been applied the on the study's dataset. The discussion is organized in subsections according to the research questions that have been introduced in the beginning of the paper. Thus, section 5.1 discusses which design patterns are more frequently used in open source software development, section 5.2 discusses the usage of each design pattern on three software categories and section 5.3 discusses crucial factors that influence the extent of pattern usage in open source software.

5.1 Design Pattern Application

The results of Table 1, clearly suggest that some patterns are more applied in open source than others. In addition to that, Table 2 suggests that pattern usage intensity classifies patterns in five categories as shown in Figure 1. In addition to that, Figure 2 represents the error bars of the eleven patterns under study.

Patterns on the top of Figure 1 are statistically significantly employed more times in open source software projects than those closer to the bottom of Figure 1.

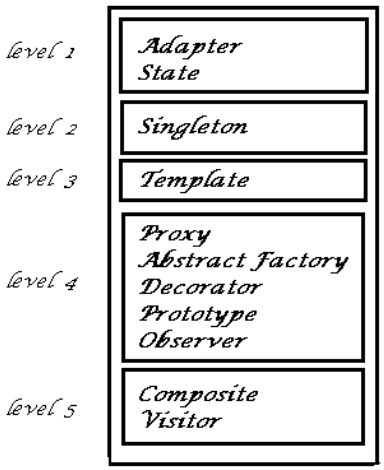

Figure 1: Design Pattern Usage Levels

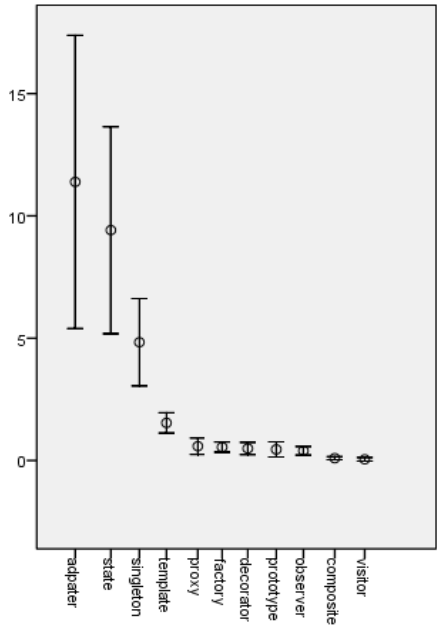

Figure 2: Design Pattern Usage Error Bars

Some of the results that are presented in Figures 1 and 2 are reasonable, whereas some findings are surprising. First of all, one would expect that the *Adapter* pattern to be frequently used, because reusing classes of others is a common practice in open source software communities. In such cases, adapter provides a mechanism for adopting the new class in the existing system without modifying the existing code. In addition to that, the Adapter's rationale is a kin to the basic concepts of object - oriented programming and thus it might be unconsciously used by the developers. Furthermore, the *State* pattern would also be expected to rank high, because its background only requires the proper use of inheritance. Finally, more difficult to understand patterns, according to authors' opinion, such as *Visitor* and *Observer*, are not often employed by open source developers

On the contrary, although the *Singleton* pattern's structure is quite complex in its structure (Chatzigeorgiou, 2005) and it was expected not to be as popular, it is ranked as the 3rd most used pattern. A possible reason for this is the limitation of the case study subject to the Java languages, where Singleton is implemented by a simple instantiation mechanism. Another bizarre observation is that the *Decorator* pattern is more frequently used than the *Composite* pattern. The Composite pattern is the base of the Decorator pattern and therefore it was expected to be more frequently employed.

Summing up the above, open source developers employ easy to understand patterns more than more elaborate ones. A possible reason for this is that typically there are no detailed design activities before programming.

5.2 Design Patterns and Software Categories

As it is observed in Table 3, design pattern usage within every category follows similar distribution as in open source software development in general. However, comparing pattern application across software categories, the results suggest that some patterns are more frequently applied in one category, than another.

From Table 4, it is suggested that *Composite* pattern is employed more frequently in OO Development tools than in the other two categories. This fact can be justified by the expectation that developers of this category are more likely to be aware of the pattern, which is not easily applied by chance. In addition to that, the *Adapter* pattern is more frequently employed in games than in OO Development tools and this fact suggest that game developers might perform more reuse activities than other programmers. This observation is interesting and deserves further investigation.

Furthermore, the *Observer* and the *Proxy* patterns are less frequently applied in e-commerce applications than games or OO Development tools. Considering the above, and by taking into account the fact that the *Visitor* and the *Composite* patterns

are not applied in any e-commerce application, it is implied that e-commerce developers more often use patterns of levels 1, 2 and 3 (*see Figure 1*) and they lag in the use of patterns of levels 4 and 5. This fact suggests that e-commerce developers might use patterns that are "easy to guess".

5.3 Factors Influencing Design Pattern Application

This section of the paper discusses the most important factors that influence design pattern application. According to Table 6, the days that a project is active, the number of downloads, the number of developers and the project size are the most influential factors of pattern usage. The number of versions also seems to be correlated to the employment of structural patterns, but concerning behavioural and creational patterns it seems not to be influential. On the other hand, the software release year and the software category do not seem to be important.

Concerning the data of Table 7, the most informative clusters are cluster [1] and [5], where the profiles of an average computer game and an e-commerce application that does not employ any pattern are fingerprinted. The centres of the two clusters are presented in Table 8.

Table 8: Average Game and E-Commerce Application without any pattern profiles

Cluster No	**[1]**	**[5]**
Release year	2006 – 2007	after 2008
Days Active	more than 2 years	less than 1 year
Downloads	2,000 – 10,000	less than 2,000
Developers	more than one	one developer
Versions	2 – 10 versions	one version
Classes	more than 200	21 – 50
Factory	1 – 4 pattern instances	no pattern instance
Prototype	no pattern instance	no pattern instance
Singleton	more than 15 pattern instances	no pattern instance
Adapter	more than 15 pattern instances	no pattern instance
Composite	no pattern instance	no pattern instance
Decorator	1 – 4 pattern instances	no pattern instance
Proxy	1 – 4 pattern instances	no pattern instance
Observer	1 – 4 pattern instances	no pattern instance
State	more than 15 pattern instances	no pattern instance
Template	5 – 9 pattern instances	no pattern instance
Visitor	no pattern instance	no pattern instance
Category	Game	E-Commerce

6. THREATS TO VALIDITY

This section of the paper deals with presenting the case study's internal and external threats to validity. Firstly, since the case study subjects have been open-source projects, the results may not apply in closed source software. Concerning the empirical study internal validity, the existence of confounding factors is analyzed in section 3.4. In addition to that, the sample size is quite small with respect to the total number of open source games and generalizing the results from the sample to the population is risky.

In addition, the dataset consisted only from Java projects, since the tool we used was able to detect design patterns only in binary java files. Moreover, only one repository, namely Sourceforge, has been mined. Additionally, the project size has been represented through the Number of Classes (NOC) metric, since patterns are class collections. An alternative would be to measure physical size, in terms of the Lines of Code (LOC) metric.

Finally, a possible threat to the validity of the results is that the project's team size takes into account the absolute number of developers and not the intensity of their activity, e.g. in terms of number of commits.

7. CONCLUSIONS

This study empirically investigates the usage of object oriented design patterns in open source software development. For this reason the authors have explored 108 open source software from three categories, i.e. object oriented development tools, e-commerce application and computer games.

The results of the study confirm that "easy to use" design patterns, such as Adapter, State and Singleton are more frequently applied in open source. More elaborate patterns such as Visitor and Observer are more frequently employed by object-oriented development tool programmers, most probably due to their better understanding and knowledge on software engineering issues. Concerning the factors that influence pattern application, the size and activity of the project, i.e. duration, downloads, the team size have proved to be the more important ones. On the contrary, the number of versions does not seem to influence pattern activity. Finally, the frequent application of the Adapter pattern in computer games might indicate higher reuse levels in this type of software applications. We intend to further investigate this

issue to understand better reuse opportunities in game development.

As future work we are about to create a web repository on the findings of the design pattern detection process, so as to enhance design pattern reuse opportunities. In addition to that, we are going to explore projects written in other programming languages, such as C++. Finally, more software categories and open source projects are going to be investigated.

REFERENCES

Ampatzoglou A., Chatzigeorgiou A., 2007 *In Information and Software Technology,* "Evaluation of object-oriented design patterns in game development", Elsevier.

Arnout K., Meyer B., 2006 *In Innovations in Systems and Software Technology,* "Pattern componentization: the factory example", Springer.

Bansiya J., Davis C., 2002, *In IEEE Transaction on Software Engineering*, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Computer Society.

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

Bieman J.M., Jain D., Yang H.J., 2001, *In ICSM 2001, 17th International Conference on Software Maintenance*, "OO design patterns, design structure, and program changes: an industrial case study", IEEE Computer Society

Chatzigeorgiou A., 2005, *"Object-Oriented Design: UML, Principles, Patterns and Heuristics"*, Kleidarithmos, Athens, 1st edition.

Di Penta M., Cerulo L., Gueheneuc Y. G., Antoniol G., *In ICSM 2008, 24th International Conference on Software Maintenance,* "An Empirical Study of Relationships between Design Pattern Roles and Class Change Proneness", IEEE Computer Society

Feller J., Fitzgerald B, 2002, *"Understanding open source software development"*, Addison-Wesley Longman, Boston, MA, 1st edition

Gamma E, Helms R, Johnson R, Vlissides J, 1995, *"Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison-Wesley Professional, Reading, MA, 1st edition.

Hsueh N.L., Chu P.H., Chu W., 2008, *In Journal of Systems and Software*, "A quantitative approach for evaluating the quality of design patterns", Elsevier.

Huston B., 2001, *In Journal of Systems and Software*, "The effects of design pattern application on metric scores", Elsevier.

Khomh F., Gueheneuc Y.G., 2008, *In CSMR 2008, 12th European Conference on Software Maintenance and Reengineering,* "Do design patterns impact software quality positively?", IEEE Computer Society

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

McShaffry M., 2003, *"Game Coding Complete"*, Paraglyph Press, Arizona.

Meyer B., Arnout K., 2006, *In IEEE Computer*, "Componentization: The Visitor Example", IEEE Computer Society

Prechelt L., Unger B., Tichy W. F., Brossler P., Votta L. G., 2001, *In IEEE Transactions on Software Engineering,* "A controlled experiment in maintenance comparing design patterns to simpler solutions", IEEE Computer Society.

Samoladas I., Stamelos I, Angelis L., Oikonomou A, 2004, *In Communications of the ACM,* "Open source software development should strive for even greater code maintainability", Association of Computing Machinery

Sowe S.K., Angelis L., Stamelos I., Manolopoulos Y., 2007, *In OSS 2007, Open Source Software Conference,* "Using Repository of Repositories (RoRs) to Study the Growth of F/OSS Projects: A Meta-Analysis Research Approach", Springer

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

Vokác M., Tichy W., Sjøberg D.I.K., Arisholm E., Aldrin M., 2003. *In Empirical Software Engineering,* "A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns - A Replication in a Real Programming Environment", Springer.

Wendorff P, 2001. *In CSMR 2001, 5th European Conference on Software Maintenance and Reengineering,* "Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project", IEEE Computer Society.

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

APPENDIX – A

Project Name	
Modus	Risk
Jupe	Summer Project gone crazy!
webMethods jasper BI	FreeRails
JUndo Runtime	Rescue! Max
OVal	Hunt for Gold
Morph	JFlag
rMock	Tourist Camping Tycoon
jfw	PokerApp
Eclipse Metrics	xUNO ME
Java Accumulators and Variants	JSkat

transmorph	Truco! o algo...
Fast UML	Jokers
VFSJFileChooser	DaCoinch
Code 2 UML	JGames
iGesture	TrickTakingGame
Java Bean Library	JMhing
Cotta: A better file system	JiBi's Hold'Em
Java Utilities	Dr. Scenario
jCart	Stigma - The Game
Jfun	Arabian Flights
UBIQLIPSE	Motherload Unlimited
Open Blackboard System	Jippy Snake
JoSQL (SQL for Java Objects)	JMario
Memories of Mordor	MazeRunner
Java Object Mapping Project	JMaze
UCDetector	YATI
JsonMarshaller	jLodeRunner
Luigi Open Search Engine	XSwing Plus
AODV Simulator	Scorched Earth 2000
im4java	Piano Sheet Music Teacher
JOpt Simple	BlacKNight Chess
Salto Persistence Mechanism	The Java VGA-Planets Client
Ajax JSP Tag Library	Three-dimensional Go Game
ftp4j	SuperSnake
GreenP UML	Gomoku
BeanForm	Marauroa
JCommons	Clippers
SAP-JCO Support	CarDriving2D
XConf	miniTraff
JRegistry	The Tao of Soccer
Perola	faceCart
JDots	FreeCol
jMagazzino	FreeLords
Broadleaf Commerce	Panzer Combat II
JadaSite	Domination
Artabro ERP/TPV	Colossus
StoreManager	opaals tools
AgEx	tXtFL
Intrannuity iBilling Client	TGame2
Jataka CRM	XHSI
NEOSMAS RentaCar	CarDriving
ShopDB	Tunneling2java
Arcus - Rubik's Cube Simulator	Shop - 123JavaShop
JSettlers2 - Java Settlers of Catan	Dioscuri - modular emulator

APPENDIX - B

Categorical Variables Thresholds

For variable (3)
0 – Object oriented development tool
1 – E commerce application
2 – Computer game

For variable (4)
1 – Released before 2005
2 – Released between 2006 - 2007
3 – Released after 2008

For variable (5)
1 – Active for less than 100 days
2 – Active for less than 1 year
3 – Active for less than 2 years
4 – Active for more than 2 years

For variable (6)
1 – Less than 1.000 downloads
2 – Less than 2.000 downloads
3 – Less than 10.000 downloads
4 – More than 10.000 downloads

For variable (8)
1 – Only one version
2 – Between 2 – 10 versions
3 – More than 10 versions

For variable (9)
1 – 21 to 50 classes
2 – 51 to 100 classes
3 – 101 to 200 classes
4 – More than 201 classes

For variables (10-22)
1 – No pattern instances
2 – 1 to 4 pattern instances
3 – 5 to 9 pattern instances
4 – 10 to 14 patterns instances
5 – More than 15 pattern instances

Binary Variables Thresholds

For variable (7)
0 – Only one developer
1 – More than one developers

For variables (10-22)
1 – The project does not have any pattern instance
2 – The project has at least one pattern instance