INVESTIGATING THE USE OF PATTERNS IN OPEN-SOURCE GAMES

APOSTOLOS AMPATZOGLOU, ALEXANDER CHATZIGEORGIOU, NIKOLAOS SAMARAS

Department of Applied Informatics, University of Macedonia
Egnatia 156, Thessaloniki, Greece
E-mail: ampatzoglou@doai.uom.gr, achat@uom.gr, samaras@uom.gr

Game development is considered to be one of the most profitable fields in the software industry. Since game production is a very complicated task to accomplish, software engineering techniques could prove useful for providing maintainability and reusability. This paper investigates to what extent open-source game developers are familiar with design patterns and apply them in order to improve games' structure. Additionally, some generic subjects on design patterns are being examined, such as pattern difficulty and relationships between design patterns of the same family.

Keywords: Games, design patterns, statistical analysis, decision trees, regression analysis

1. INTRODUCTION

It goes without saying that computer game design is one of the most modern and fast growing trends in computer science (T.M. Rhyne, 2000, pp. 519-521). Until the middle of 90's, game developers did not aim to produce reusable code since every program has been written from scratch in assembly language (Rollings, 2003). Later, reusable coding has proven to be one of the most important issues in game development because games became far more complex and their production process more time consuming. In order to alleviate this problem, frameworks and game engines have been created. A framework is a collection of classes that can be widely reused and integrated with other components (R. Rucker, 2003) and (L. Valente, 2005). Usually they implement mechanisms that occur in many games, such as input handling, file handling (texture, models, audio etc), 3D rendering etc. Game engines are programs that provide developers the potential to design game levels, handle player and opposition behavior, by using scripting languages and powerful GUIs. As it is easily understood, if frameworks and game engines are "well-structured", they can be maintained without extreme effort and be transformed so that they can fit as many game genres as possible.

The application of design patterns in systems (C. Alexander, 1977) is a quite popular software engineering technique. Employing patterns in software design and implementation is considered to provide code reusability and maintainability (E. Gamma, 1995). The benefits and drawbacks of the application of patterns in classical programming (not game related) are discussed in a few papers (L. Prechelt, 2001, pp.1134-1144) and (M. Vokác, 2004, pp. 149-195). The results of applying design patterns in games, concerning complexity, cohesion, coupling and size have also been examined (A. Ampatzoglou, 2007, pp 445-454). The results of the aforementioned research imply that patterns reduce complexity of games, increase cohesion between methods, reduce coupling but increase the size of the examined games.

This paper aims at investigating whether open-source game developers are familiar with patterns and if they actually employ them. Furthermore, it examines whether game developers are familiar with patterns of the same family and design pattern comprehension difficulty. The results have been obtained through an email survey that was conducted with

questionnaires sent to open-source game programmers. The feedback from the questionnaires was analyzed with statistical and data mining packages

In the literature there are several efforts concerning statistical analysis on the use of patterns in classical programming. More specifically, Prechelt in (L. Prechelt, 1997), asked the programmers participating in an experiment to fill in a questionnaire in order to identify their background in design patterns. The subjects had an average of 5 years of programming experience and 91% of them had written more than 3000 Lines Of Code (LOC) in their life. The most known pattern between them proved to be Iterator and the least known, the Proxy. Similarly to investigating design patterns for game development, in (J. Borchers, 2002) the author examined the usefulness of Human-Computer Interaction (HCI) patterns. In his survey the author asked students to fill in questionnaires, on the completion of a course. The results suggested that each student remembered an average of 1.73 patterns, rated the usefulness for learning 1.96 (out of 5), the usefulness for current project 2.23 (out of 5) and the usefulness for reuse in future projects 1.94 (out of 5).

2. ANALYSIS DESCRIPTION

In order to investigate the target points of our survey, we contacted open-source game developers by e-mail and asked them to fill in a questionnaire[*]. Their addresses were most commonly found from a wide open-source community (Sourceforge) and from universities that offer undergraduate or postgraduate courses in game programming. In total, 245 e-mails have been sent and 29 questionnaires have been received (response rate=11.8%, failure rate=2.1% (K.L. Manfreda, 2005). The methodology of the survey was based on (K.L. Manfreda, 2005) that suggested to provide individual invitations to any possible target and to avoid open-ended questions.

The questionnaire has been divided into two subsections. The first includes information about the programmer such as experience (in years) in the most common programming languages and whether he is familiar with some design patterns. The second section included information about a certain game indicated by the developer. More specifically, we retrieved information about the dimensionality of the game, the packages used for 3D graphics, the appearance of design patterns in it and additional information about the overall belief in the usefulness of patterns. The complete questionnaire led to the construction of the database. The questionnaire consists of 10 questions from which we extracted 94 variables. Most of the extracted attributes were either binary or numeric. Some of those variables are coded into a three category scale.

The attributes extracted from the questionnaires have been analyzed using two intersecting scientific approaches, statistics and data mining. This kind of analysis is quite popular in scientific research as shown in (G. Alexe, 2005, pp.322-325), (R.S. Lin, 2006) and (D.J. Newman, 1998). Both techniques can be combined as follows; statistical tests can be applied to help analyze the results of a data mining session. Statistical analysis is clearer in its goal, so it is not easy to face an unexpected result, and the processing includes a beginning and an ending. On the other hand the purpose of a data mining session is to identify hidden and potentially useful knowledge from a data set. The knowledge gained is given as a conceptual generalization of the data set. Hidden knowledge can be mined using supervised learning and/or unsupervised clustering. Decision or classification trees are one of the most popular tools for constructing classification models in the software engineering field (Khoshgoftaar, M.T., 1999) and (Tian, J., 1998, pp. 97-104). Decision trees are induction techniques used to discover the interactions among predictors that do not exhibit strong marginal effects (N.R. Cook, 2004, pp.1439-1453). Roughly speaking, this can be achieved by subdividing the information contained in the data set.

[*] The questionnaire can be found at http://java.uom.gr/~apamp/publications.html

2.1 DATA MINING

As mentioned above data mining techniques are used in this survey in order to identify some indication for related data. So, the selected technique should generate some kind of association between fields of the dataset and provide measures that will declare how strong the connection is. In this survey we preferred to use decision trees and more specifically a tool (Saha) that is available in the web. This tool uses the C4.5 algorithm (Quinlan, J.R., 1993) for building a decision tree.

Building a decision tree requires an input data set, known as training data that can be composed of either continuous/discrete (numeric) or categorical (concrete states) fields. The output attribute (class) must obligatory be of categorical value. Consequently, if somebody wants to predict a numeric field he must transform his data in order to convert continuous fields to categorical. C4.5 is initialized by selecting a subset (T1) of instances from a training set. This subset used by the algorithm to build a decision tree. The remaining instances (T2) test the validation of the built decision tree. At every iteration, C4.5 selects the attribute best able to represent the largest amount of gain in information. The used tool (Saha) after the completion of growing the tree generates a set of production rules. Production rules are stated as "IF <CONDITION(S)> THEN <RESULTS>", where <CONDITION(S)> stands for one or more input attributes and <RESULTS> stands for the value of the output attribute (class).

In order to validate the extracted rules the tool calculates *support, confidence* and *capture* for every rule that it generates. The support value declares how widely applicable the rule is. The confidence value measures the accuracy of the rule. *Capture* declares the percentage of <*RESULTS*> that the rule classified correctly. In order to provide a measurement that generally describes the strength of the rule by taking into account both support and confidence, we calculate the *lift* measurement.

$$Support(Conditions \Rightarrow Results) = \frac{\#\text{ of records containing (Cond, Res)}}{total\ \#\ \text{of records}} \quad (1)$$

$$Confidence(Conditions \Rightarrow Results) = \frac{\#\text{ of records containing (Cond, Res)}}{\#\ \text{of records containing (Cond)}} \quad (2)$$

$$Lift(Conditions \Rightarrow Results) = \frac{\text{Confidence (Cond} \Rightarrow \text{Res)}}{\text{Support(Res)}} \quad (3)$$

2.2 STATISTICAL ANALYSIS

The statistical analysis of the dataset derived from questionnaire results can be divided into two subcategories, descriptive statistics to summarize the dataset under study and regression analysis to determine the magnitude of the relationships between the examined variables.

The first part is quite common and it is not necessary to describe the statistical measures used, so they are just referenced by name. The analysis has been conducted using frequency, average value, standard deviation, variance and graphs (M. Norusis, 2005). The target attributes of the analysis will be object-oriented experience, familiarity with patterns and application of patterns in games.

The second part is the regression analysis that aims in identifying correlated attributes and provide an estimate of the magnitude of the relationship between them (M. Norusis, 2005). The regression analysis methods that have been used in the survey are two: logistic regression and multinomial logistic regression. The most famous regression type is the linear regression that estimates the coefficients of the linear equation, involving one or more

independent variables that best predict the value of the dependent variable. Logistic regression is similar to a linear regression model but is suited to models where the dependent variable is dichotomous. Finally, multinomial logistic regression is useful for classifying subjects based on values of a set of predictor variables. This type of regression is similar to logistic regression, but it is more general because the dependent variable is not restricted to two categories (M. Norusis, 2005). The target fields for this analysis type will be mentioned by the end of section 3.2.2 where the results of classification will be presented.

3. RESULTS

As mentioned above (section 2) we conducted a two-step analysis. The investigation aimed at exploring the connection of understanding design patterns within the same family, the difficulty of understanding each pattern and the possible connection between applying patterns in games, performing design activities, using game engines and object-oriented programming experience.

3.1 DESCRIPTIVE STATISTICS RESULTS

The results of the descriptive statistical analysis will be presented through graphs and tables. The object-oriented experience is described in the dataset by two attributes, one categorical (*HIGH-EXPERIENCE*, *MEDIUM-EXPERIENCE* and *LOW-EXPERIENCE*) and one numerical (reflects the exact years of object-oriented programming experience). In the same way the experience of developers with patterns and the actual pattern application in a specific game are reflected in other four attributes of the dataset.

The descriptive statistics extracted from processing the numeric attributes of the aforementioned characteristics are shown in Table 1, while Table 2 summarizes frequency statistics on the corresponding categorical variables.

TABLE 1. DESCRIPTIVE STATISTICS

	Mean		Std. Deviation	Variance
	Statistic	Std. Error		
Experience in OO programming	6.46	0.70	3.726	13.888
Number of design patterns familiar with	5.24	0.91	4.918	24.190
Number of design patterns applied	2.96	0.64	3.479	12.106

TABLE 2. FREQUENCY STATISTICS

Topic	Category	Frequency	Percentage
OO Experience	*HIGH*	8	27.6%
	MEDIUM	12	41.4%
	LOW	8	27.6%
Pattern Familiarity	*EXPERT*	9	31.9%
	QUITE-SKILLED	13	44.8%
	NOT-FAMILIAR	7	24.1%
Pattern Application	*MANY-PATTERNS*	5	17.2%
	SOME-PATTERNS	13	44.8%
	NONE-PATTERNS	11	37.9%

Taking into account the data from Tables 1 and 2, we can understand that the sample of programmers is well balanced with respect to object-oriented experience and the results about the percentage of developers being familiar to patterns and consequently the actual application of them are as close to reality as possible, as well as the associations that might be extracted. The results on pattern familiarity and actual pattern application show that even though a quite high percentage is considered *PATTERN-EXPERTS* (31.9%), the percentage of *MANY-PATTERNS* games stands for only the 17.2% of the whole dataset.

Another interesting statistic is the frequency of individual pattern knowledge and the frequency of their application. This measurement is presented as the percentage of the developers that know each pattern and used it. Additionally quite useful, in order to estimate the usefulness of each pattern, is to calculate the percentage of developers that used a pattern among the developers that were familiar with it. The results are presented in Figure 1 and Table 3.

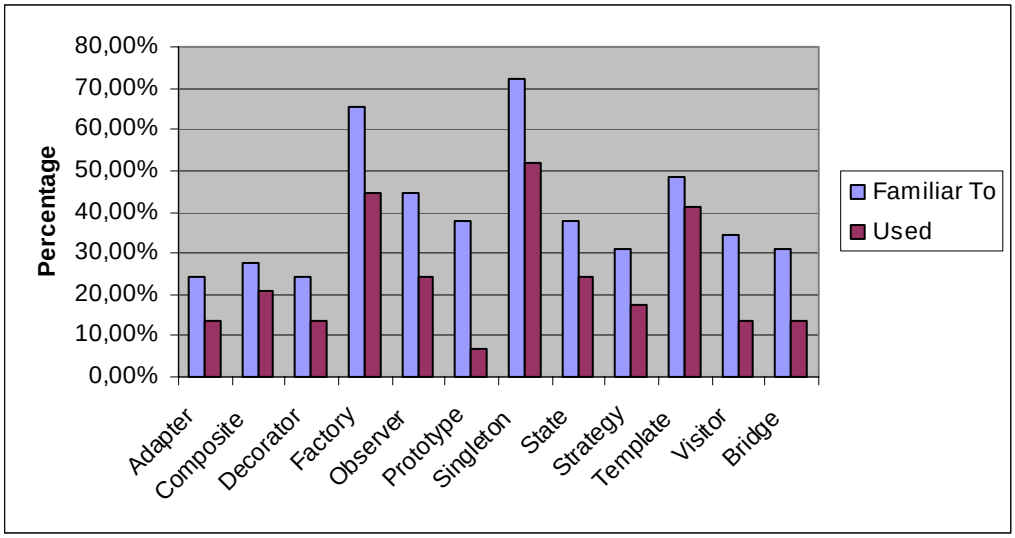


FIGURE 1. PATTERN KNOWLEDGE AND APPLICATIONS

The above diagram shows that the actual ranking of patterns with respect to familiarity and application frequency is almost identical. The only pattern that is obviously ranked in a lower position concerning application frequency rather than familiarity is the *Prototype* (ranked as 5th in familiarity and 12th in application frequency). This fact might imply that *Prototype* is not a very applicable pattern for games since only the 18.1% of those who knew it, used it.

TABLE 3. PERCENTAGE OF APPLICABILITY OF PATTERNS

Design Pattern	Frequency of Application	Design Pattern	Frequency of Application
Adapter	57.1 %	Prototype	18.1 %
Composite	75.0 %	Singleton	71.4 %
Decorator	57.1 %	State	63.6 %
Factory	68.4 %	Strategy	55.5 %
Observer	53.8 %	Template Method	85.7 %
Bridge	44.4 %	Visitor	40.0 %

Table 3 suggests that some patterns might be extremely useful in game development since they are actually used by a high percentage of developers that know them. An example of such a pattern is the *Template Method* that even though it is the 3rd most known pattern is the most applicable one, in the sense that 85.7% of developers that knew it, considered beneficial to implement at least an instance of it in a game. Similar levels of applicability are observed in *Singleton* and *Composite*. More specifically, the *Singleton* pattern appears to be suitable for game development as it is rated 1st in the actual applications (Figure 1) and 3rd in the percentage of application among the developers familiar to it (Table 3).

Moreover, at this point it is interesting to compare the results of the survey mentioned in the introduction (L. Prechelt, 1997), with those extracted for game programmers. For the six common design patterns in the two surveys the results are shown in Figure 2.

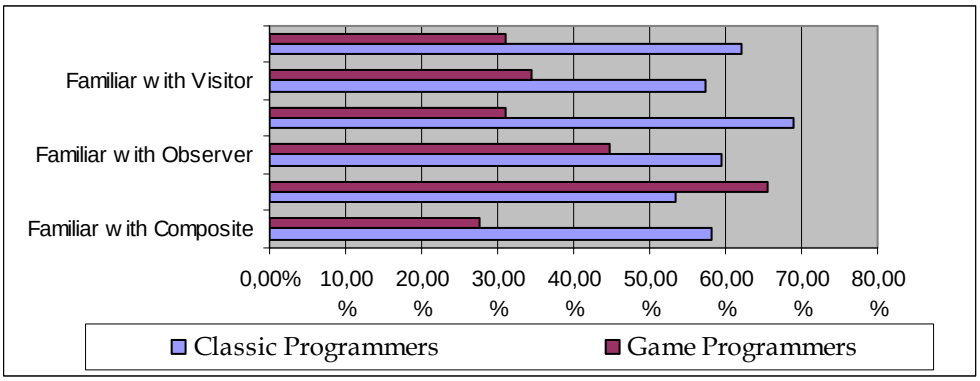


FIGURE 2. COMPARISON ON PATTERN KNOWLEDGE

Figure 2, clearly implies that game developers even if they are extremely skilled programmers, they are not as familiar as expected with design patterns. This is suggested because in 5 out of 6 (83.3 %) patterns under study their knowledge percentage is below the corresponding percentage for classical programmers.

3.2 RESULTS ON DESIGN PATTERN UNDERSTANDING

In order to investigate the connection of understanding design patterns within the same family and the difficulty of understanding patterns we employed both data mining and statistical analysis as presented in sections 3.2.1 and 3.2.2 respectively.

3.2.1 PATTERN KNOWLEDGE DEPENDENCY GRAPH

The aim of creating the pattern knowledge dependencies graph is to identify whether game developers are familiar with patterns in accordance to their categories described in (E.Gamma, 1995) and whether the knowledge of a specific pattern implies familiarity with other patterns as well. In that sense the graph will answer to questions like: "if someone is/is not familiar with *Factory* (a creational pattern) is he more likely to be/not to be familiar with *Singleton* and *Prototype* (other creational patterns) than with any other pattern?"

The relations are depicted in the graph as edges that begin from the *CONDITION* and end in the *RESULT* of the rule. The rules were firstly filtered by support, and only rules with support>=40% were selected. Afterwards, the remaining rules were sorted by lift (Rules with lift <=1 are not included in the graph)[*]. Patterns are represented as nodes in the graph (Figure 3).

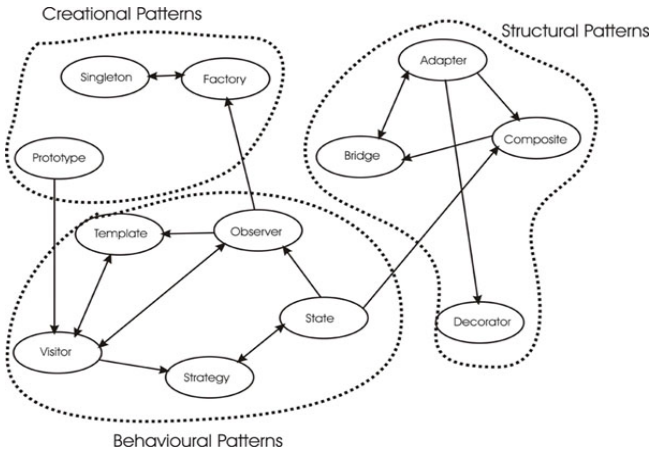


FIGURE 3. PATTERN KNOWLEDGE DEPENDENCY GRAPH

[*] The complete list of the rules generated can be found at http://java.uom.gr/~apamp/publications.html

From the graph in Figure 3 it is obvious that pattern knowledge is related to the families described in (E. Gamma, 1995). In order to quantify the dependencies between patterns of the same family we calculated the percent of edges within the same family in respect to the total number of edges. The edges that leave from nodes of a certain family and end to another node of the same family are referenced as intra-leaving edges (Intra l.e.). The edges that leave from nodes of a certain family and end in a node of a different family are referenced as inter-leaving edges (Inter l.e.). The percentage of overall intra-leaving edges in the graph is 83,3 %. In the table below (Table 5) the intra-leaving edge percentage for every pattern family is presented.

TABLE 5. INTER AND INTRA LEAVING EDGES OF PATTERNS

Pattern	Intra l.e.	Inter l.e.	Percent
Creational	2	1	66.6 %
Structural	5	0	100.0 %
Behavioural	6	2	75.0 %
Total	**15**	**3**	**83.3 %**

3.2.2 PATTERN UNDERSTANDING DIFFICULTY

In order to estimate the difficulty of understanding each pattern we used the results of two multiple multinomial regressions. The factors that might imply the difficulty of understanding a pattern are object-oriented experience and pattern knowledge, in the sense that someone extremely immature with the idea of patterns will also be familiar with the most understandable ones, similarly to someone with low experience in programming. In contrast to that, a *PATTERN-EXPERT* will be familiar with more complicated structures, similarly to a *HIGH-EXPERIENCE* programmer. In the first regression (R1), we chose Object-Oriented Experience (categorical) as dependent variable and if someone is familiar with each (totally 12) pattern (1=familiar, 0=not familiar) as independent variables. The second regression (R2) included the pattern experience (categorical) as the dependent variable, and if someone is familiar with each pattern (1=familiar, 0=not familiar) as independent variables. The results are presented below in Table 6. The average R^2 value is 0.938 that is extremely satisfying. The significance (Sig.) values for the two regressions are 0.01 and 0.00 that are statistically accepted (Sig<0.05).

Since the two regressions shared the same set of dependent variables there was no need for standardizing the coefficients. In Table 6 the Beta column represents the coefficient with which the attribute contributes to the prediction of the dependent variable and the normalized beta (n_beta) is the same value scaled in order to represent the estimated difficulty in a 0-1 scale (1=the easiest, 0=the hardest). The overall difficulty level (dl) is calculated as shown below (as an average):

$$dl = 1 - \frac{n\_beta(R1) + n\_beta(R2)}{2} \tag{4}$$

The physical explanation of the coefficient values is that as higher the coefficient for each variable is, it is more possible for a developer that is not familiar with the pattern to be a *HIGH-EXPERIENCE* programmer or a *PATTERN-EXPERT*, respectively. The negative values suggest that if someone is not familiar with a certain pattern he is not likely to be a *HIGH-EXPERIENCE* programmer or a *PATTERN-EXPERT*. For example, if the coefficient of a pattern in the prediction of object-oriented experience is less than another, then the first pattern is more probably known by more experienced programmers.

TABLE 6. PATTERN UNDERSTANDING DIFFICULTY

	Beta (R1)	N_beta (R1)	Beta (R2)	n_beta (R2)	Dl
Strategy	0.643	0.37	2.969	0.57	0.530
Template Method	16.397	0.48	12.213	0.70	0.410
Bridge	-18.035	0.24	33.410	1.00	0.380
Factory	92.329	1.00	-2.571	0.50	0.250
Singleton	-17.599	0.24	-38.018	0.00	0.880
Visitor	-18.175	0.24	-36.716	0.02	0.870
Composite	-39.840	0.09	-25.447	0.18	0.865
Decorator	-35.626	0.12	-21.700	0.23	0.825
Observer	-53.074	0.00	-9.892	0.39	0.805
Prototype	19.827	0.50	-23.453	0.20	0.650
State	1.595	0.38	-0.630	0.52	0.550
Adapter	38.717	0.63	-18.178	0.28	0.545

The results on Table 6 indicate that some patterns, like *Template Method*, *Bridge* and *Factory* are widely known among developers with low object-oriented experience or patterns familiarity. This fact clearly implies that these patterns are more easily understood than others. On the other hand, *Singleton*, *Composite*, *Visitor*, *Decorator* and Observer can be considered more complex since developers that are familiar to them are either extremely experienced in object-oriented programming or in design patterns. The results of the table above and the composition of the pattern knowledge graph in section 3.2.1 might prove interesting in the way that patterns should be referenced in a corresponding course.

3.3 FACTORS INFLUENCE DESIGN PATTERN APPLICATION

In this section it is attempted to identify which attribute is more closely related to pattern application in open-source games. The candidates examined are game engines use, object-oriented experience and design activities. The method used is backward stepwise LR binary regression. In order to perform binary regression the pattern application has to be represented by a dichotomous attribute. In order to achieve this expectation we have created two new variables for every developer's game, *HIGH-PATTERN* games (patterns_applied > 7.5) and *LOW-PATTERN* games (patterns_applied < 7.5).

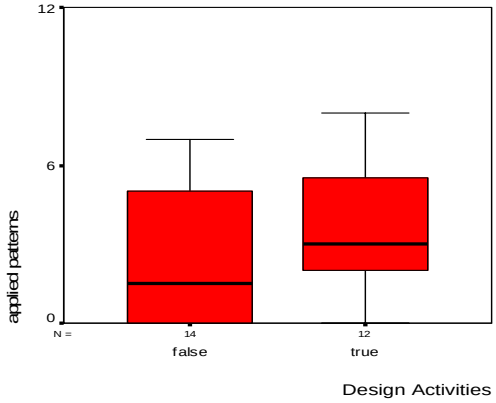
The correct classification for the model is 92.3%. The results are presented in the table below (Table 7). More specifically it is implied that the most significant factor for pattern application is design activities, followed by object-oriented experience.

TABLE 7. BACKWARD LR LOGISTIC REGRESSION FOR PATTERN APPLICATION

		Beta	Std.Err.	Sig.
Step 1(a)	Design Activities	3.134	1.969	0.111
	Game Engines	-0.839	1.449	0.563
	OO Experience	0.585	0.303	0.053
	Constant	-7.153	3.360	0.033
Step 2(a)	Design Activities	3.127	1.950	0.109
	OO Experience	0.551	0.286	0.054
	Constant	-7.103	3.296	0.031

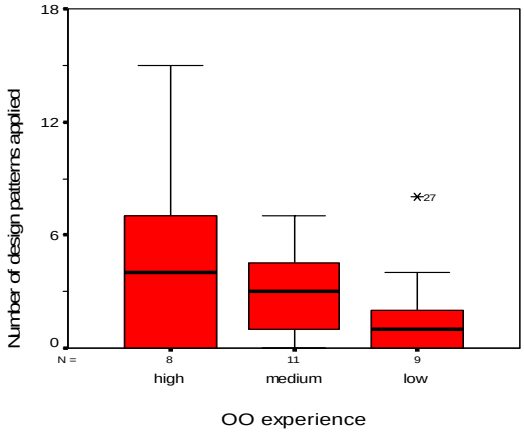
In order to strengthen the above statement we will present a boxplot that shows the relationship between design activities and the numeric attribute that represents the number of patterns that is applied in each game. The boxplot is presented in Figure 4, below.

FIGURE 4. BOXPLOT DEPICTING THE RELATIONSHIP BETWEEN DESIGN ACTIVITIES AND PATTERN APPLICATION



From the boxplot it is obvious that developers that execute design activities before coding are more likely to apply more patterns in the game they create. Although this claim is quite clear, the difference is not as large as it might be expected. The same methodology has also been used in investigating the relation between object-oriented experience and pattern application. The corresponding boxplot is presented in Figure 5.

FIGURE 5. BOXPLOT DEPICTING THE RELATIONSHIP BETWEEN OBJECT-ORIENTED EXPERIENCE AND PATTERN APPLICATION



The above figure shows that *MANY-PATTERN* games are usually developed by *HIGH-EXPERIENCE* programmers. But the majority of *HIGH-EXPERIENCE* programmers tend to implement *SOME-PATTERNS* games, similar to the *MEDIUM-EXPERIENCE* programmers (bold line of the boxplot in Figure 6). In order to further investigate why *HIGH-EXPERIENCED* programmers sometimes create *NONE-PATTERNS* games we tried to identify a relation between object-oriented experience and design activities. Performing data mining analysis, the production rules implied that 75% of *HIGH-EXPERIENCE* programmers do not perform design activities before programming the game. This fact itself partly explains why some *HIGH-EXPERIENCE* programmers sometimes create *NONE-PATTERNS* games according to Table 7.

3.4 DISCUSSION

The survey aimed at investigating some target points concerning game developers and design patterns. The results of the survey can be divided and presented into two categories: results on individual patterns and results on possible correlation of object-oriented experience, design activities, game engines and design patterns. Some of these results comply with common sense, but others are quite strange and need further investigation.

First of all, game developers appeared to be familiar with design patterns at a lower degree than classical programmers (Figure 3). More specifically, the only pattern that was more popular among game developers rather than other programmers proved to be *Factory*. Additionally, some patterns which might be more applicable in game development have been identified (Table 3). Furthermore, pattern knowledge proved to be associated with pattern families described in (E. Gamma, 1995) as shown in (Figure 4). The connection proved to be stronger in *Structural* patterns and less strong in *Creational* patterns. Moreover, it has been attempted to calculate the difficulty of understanding a pattern (Table 6). The results suggested that clearly some patterns are more difficult to understand than others. For example, the *Visitor* pattern, which is widely acknowledged as complicated due to the use of dual dispatch ranks as the second most difficult pattern. The *Singleton* pattern, although simple in its static structure, requires detailed knowledge of the corresponding code implementation (private constructor, static self-reference and static get method), something which possibly explains the estimated difficulty. On the other hand, the group *State/Strategy/Template Method* patterns which essentially focus only on the use of abstraction are considered quite easy to understand.

Concerning the use of patterns in game development and the factors that could be connected to it, design activities proved to be the most influential (Table 7, Figure 5). According to the survey if someone does not perform design activities before programming he will probably create a game that does not implement any instance of a pattern. Additionally, object-oriented experience has proven to be a minor factor for applying patterns to games (Table 7, Figure 6).

The target group of the survey was limited to open-source game programmers, so there can be no conclusions for industrial game development. Therefore, it is quite risky to generalize the conclusions extracted.

4. CONCLUSIONS

This paper aimed at investigating the use of object-oriented design patterns in open-source game development. In order to achieve the survey's goals we conducted an email-based survey. The target group of the survey was open-source game developers that have been asked to fill-in a questionnaire. The dataset extracted from the responds was processed using data mining and statistical analysis.

The results of the survey implied that game developers are less familiar with design patterns than other programmers; that some patterns are more appropriate for game programming and that design activities are the most significant factor for creating pattern-based games. An unexpected finding was that developers with high degree of experience in object-oriented programming tend to create games with a limited number of patterns in a similar rate to the one of low experience programmers. This might occur because high experience programmers tend not to execute design activities before coding.

In any case, we adopt the unanimous opinion of the pattern experts of our survey, that the application of design patterns in games certainly improves their extensibility.

REFERENCES

1. C. Alexander, S. Ishikawa, M. Silverstein, *A Pattern Language – Town, Buildings, Construction*, Oxford University Press, New York, 1977
2. G. Alexe et.al, "A robust meta-classification strategy for cancer diagnosis from gene expression data", *Procedures of IEEE Computer Systems Conference,* 322-325, 2005
3. A. Ampatzoglou, A. Chatzigeorgiou, "Evaluation of object-oriented design patterns in game development", *Information and Software Technology,* Vol. 49, Issue 5, May 2007, pp 445-454

4. J. Borchers, "Teaching HCI Design Patterns", *Patterns in Practice: A Workshop for UI Designers" workshop at CHI 2002 International Conference on Human Factors of Computer Systems*, Minneapolis, MI, April 21-25, 2002
5. N.R. Cook, R.Y. Zee, P.M. Ridker, "Tree and spline based association analysis of gene-gene interaction models for ischemic stroke", *Stat Med*, 23. 1439-1453, 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. D. J. Hand, "Statistics and Data Mining: Intersecting Disciplines", *ACM SIGKDD*, June 1999, volume 1, issue 1, pages 16-19
8. Khoshgoftaar, M.T. and Allen, B.E. (1999). "*Modeling software quality with classification trees", In Recent Advances in Reliability and Quality Engineering"*, Hoang, P. (Ed.), World Scientific.
9. R.S. Lin, S.Y. Rhee, R.W. Shafer, A.K. Das, "A combined data mining approach for infrequent events: Analyzing HIV mutation changes based on treatment history", *Computational Systems Bioinformatics*, Stanford, 2006.
10. K.L. Manfreda, V.Vehovar, "Survey Design Features influencing rates in web surveys", *International Conference on Improving Surveys*. Copenhagen, October 2005 http://www.icis.dk/ICIS_papers/C2_4_3.pdf
11. D.J. Newman, S. Hettich, C.L. Blake, C.J. Merz, "UCI Repository of machine learning databases", 1998, http://www.ics.uci.edu/~mlearn/MLRepository.html
12. M. Norusis, *"SPSS 13.0 Guide to data analysis"*, Prentice Hall (Feb 2005)
13. L. Prechelt, B. Unager, D.C. Schmit, "Replication of the first controlled experiment on the usefulness of design patterns: Detailed description and evaluation", Technical Report wucs-97-34, December 1997, http://www.cs.wustl.edu/cs/techreports/1997
14. L. Prechelt, B. Unger, W. F. Tichy, P. Brossler, L. G. Votta, "A controlled experiment in maintenance comparing design patterns to simpler solutions.", *IEEE Transactions on Software Engineering* 27(12), pages 1134-1144, 2001
15. Quinlan, J.R. (1993). *"Programs for machine learning"*, San Mateo, CA: Morgan Kaufmann.
16. 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
17. Rollings, D. Morris, *"Game Architecture and Design: A New Edition"*, New Riders, Indianapolis, 2003.
18. R. Rucker, *"Software engineering and computer games"*, Addison Wesley, Essex, United Kingdom, 2003
19. Saha, "Classification Tree in Excel", http://www.geocities.com/adotsaha/CTree/CTree.zip
20. Sourceforge.net, open-source community, http://www.sourceforge.net
21. Tian, J. and Palma, J. (1998). "Analyzing and improving reliability: A tree-based approach". *IEEE Software*, 15(2), pp. 97-104.
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. Vokác, W. Tichy, D.I.K. Sjøberg, E. Arisholm, M. Aldrin, "A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns - A Replication in a Real Programming Environment", *Empirical Software Engineering*, vol. 9, pages 149-195, 2004