Design Pattern Alternatives: What to do when a GoF pattern fails

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

Sofia Charalampidou
Department of Computer Science
Chalmers University of Technology
Gothenburg, Sweden
sofiagcharalampidou@gmail.com

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

ABSTRACT
Design patterns have been introduced in the field of software engineering in the middle of 90s as common solutions to common design problems. Until now, the effect of design patterns on software quality attributes has been studied by many researchers. However, the results are not the expected ones, in the sense that several studies suggest that there are cases when a design pattern is not the optimum way of designing a system. In this paper, we present the findings of a systematic literature review that aims at cataloging published design solutions, referenced as alternative design solutions, which are equivalent to design patterns and can be used when a design pattern instance is not the optimum design solution for a specific design problem.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**]: Design Tools and Techniques - *Object-oriented design methods*. ACM Computing Classification Scheme: http://www.acm.org/class/1998/

General Terms
Design

Keywords
design patterns, design alternatives, literature review.

1. INTRODUCTION

One the most famous software pattern catalogues has been introduced by Gamma, Helms, Johnson and Vlissides, where the authors catalogued 23 design solutions to common design problems in the area of object-oriented software engineering, known as Gang of Four (GoF) design patterns [11]. Since then, many papers evaluated GoF design patterns with respect to quality attributes, by comparing them to alternative design solutions, sometimes referenced as "simpler solutions".

PCI 2013, September 19 - 21 2013, Thessaloniki, Greece

http://dx.doi.org/10.1145/2491845.2491857

In [5], the authors synthesize the findings of the abovementioned papers, in a mapping study, that reports on the effect of design patterns on software quality attributes. The results of [5] suggest that the effect of GoF patterns on software quality is not always positive, but there are cases when an alternative design solution might produce better results, with respect to certain quality attributes. In addition to that Wendorff presents a case study conducted in an industrial environment, where GoF design pattern have been inappropriately applied. As a result, the maintainability of the system was decreased [34]. Finally, in [4], the authors suggest that even the same design pattern can have a different effect on the same quality attribute, depending on the number of classes that participate in the design pattern instance.

In this study, we conduct a systematic literature review that aims at cataloguing a broad range of design solutions that are equivalent to GoF design patterns. At this point it is necessary to clarify that the recorded alternatives are not necessarily "good" or "bad" practices, in the sense that in most cases, these alternatives have not been compared to the GoF design pattern they substitute. In addition to that, the paper itself does not aim to compare the design patterns and the design alternatives, as it is done on some of the primary studies and summarized in [5]. However, whenever possible, the paper provides pointers to primary studies that whether compare patterns to alternative solutions.

Considering the above, it becomes clear that a list of alternative design solutions, which are equivalent to GoF design patterns can be interesting for both researchers and practitioners. Such a catalogue would help practitioners to pick a design solution, in cases when GoF design patterns are applicable, but they are not the optimum design solution, with respect to the software quality attributes that they are interested in. Concerning researchers, such a catalogue is expected to aid in the evaluation of GoF design patterns, with empirical and analytical methods, such as in [3, 4, 25 and 31]. In section 2, we describe the methodology that is used during the review process. Section 3 presents the results of our secondary study, whereas section 4 discusses them.

2. METHODOLOGY

In order to accomplish a high standard systematic literature review process, we have chosen to follow the guidelines described in [17], which are considered the leading methodology for conducting and presenting systematic reviews in software engineering.

2.1 Research Questions

In this study, we plan to investigate the alternative design solutions that are equivalent to GoF patterns. The main research questions addressed in the study are:

***RQ_1**: Which GoF design patterns have equivalent alternative design solutions?*

The findings of this research question are closer to the results of a mapping study, since pointers to other primary studies are given. Thus, they will aid researchers in the sense that they will provide a catalogue of pattern alternatives and pattern variations that can be used for evaluating design patterns. On the other hand, practitioners can assess each primary study and decide if they want to use GoF pattern alternatives, without searching the complete pattern literature.

***RQ_2**: How are alternative design solutions represented in the studies they are introduced?*

This research question aims at investigating the way that such alternatives are presented. Design pattern alternatives can be described by UML class diagrams, source code or by a textual description. Intuitively, it is expected that design alternatives that are described in a more formal way, such as class diagram are easier to use and adapt.

***RQ_3**: Is there a comparison of alternative design solutions to the GoF design pattern that they are equivalent to?*

This research question aims at investigating if the proposed design alternatives are evaluated and compared to the design pattern they can substitute. The results of this research question are important mainly to practitioners who are interested in the structural quality of their final product.

2.2 Search Process

The search process of our study has been based on the process described in [7], where the authors used seven journals and seven conferences as search space. In our study, we searched in the journals and conferences of [7] and in two additional journals, seven additional conferences and two additional workshops that deal with pattern languages, reverse engineering, maintenance, refactoring, metrics, and generic software engineering[1]. Concerning the time period of the searching process, the study has not defined any starting search date and includes articles published until the end of 2010.

[1] SIGSOFT Symposium on Foundation of Software Engineering (FSE), International Symposium on Software Reliability Engineering (ISSRE), Empirical Software Engineering (ESE), Metrics Symposium (METRICS), Symposium on Empirical Software Engineering and Measurement (ESEM), International Conference on Program Comprehension (ICPC), International Conference on Programming Languages of Patterns (PLOP), European Conference on Programming Languages of Patterns (EuroPLOP), Object Oriented Programming, Systems, Languages & Applications (OOPSLA), International Conference on Automated Software Engineering (ASE), Software (IEEESoft), Journal of Systems and Software (JSS), Information and Software Technology (IST), IEEE Transactions on Software Engineering (TSE), Working Conference on Reverse Engineering (WCRE), ICSE Workshops , International Conference on Software Maintenance (ICSM), International Conference on Software Engineering (ICSE), European Conference on Software Maintenance and Reengineering (CSMR), Annual Computer Software and Application Conference (COMPSAC), FSE Workshops, Journal of Software: Evolution and Process, ACM Transactions on Software Engineering and Methodology (TOSEM), Science of Computer Programming (SCP), International Symposium in Software Testing and Analysis (ISSTA), Software Testing, Verification and Reliability (STVR).

The search process was conducted by a search through the portals of five digital libraries, namely ACM, IEEE, ScienceDirect, Springer, and Wiley. The only term used in the search process was pattern, referenced in the title of the publication. The exclusion of non-relevant articles was manually conducted according to the inclusion and exclusion criteria defined in Section 2.3.

2.3 Article Inclusion and Exclusion Criteria

The papers that are selected as primary studies in the review must present a design solution that is equivalent to a GoF design pattern [11]. In line with [17], there are three stages of filtering the article set to produce the primary study data set, i.e. on basis of title, abstract and full text. The search process was handled by the second author, the titles and abstracts have been examined by the first author whereas the full papers, which were not rejected at the first two stages, were examined by all three authors. The two inclusion criteria are the relevance of a primary study with GoF design patterns and the existence of a description of another design that is equivalent to one GoF design pattern.

In cases that more than one author was responsible for the inclusion/exclusion phase, the evaluation of the primary study was done separately. The final decision on including or excluding a study was made through agreement of all authors. The most common reason for excluding a paper, with respect to title, was that the paper dealt with patterns in topics other than software engineering. In addition, when we examined the abstracts of the papers, the majority of the excluded studies dealt with HCI patterns or architectural patterns. Finally, the criterion taken into account while excluding paper, with respect to their full text, was the absence of an explicit reference to at least one of the 23 GoF patterns in the article full text or that the paper did not have any kind of description of a design pattern alternative.

On the completion of the above process our primary study dataset included 28 papers. The journals and conferences where relevant papers have been identified, are presented in Table 1, accompanied by the number of papers that have been taken into account from every venue.

Table 1. Publication Venues[1]

Name	count
PLOP	18
JSS	2
ICSE	1
ISSRE	1
ICSE Workshops	1
TSE	1
IST	1
COMPSAC	1
EuroPLOP	1
ESE	1

2.4 Data Collection

During the article selection phase, we have collected a set of variables that describe each primary study. For every study the following data have been extracted:

- Published in (journal or conference name)

- Patterns Investigated (name of GoF pattern)
- Alternative Design Solutions to a GoF Design Pattern (the name of the design alternative if it is another design pattern. If the alternative design is an unnamed variation or design, a description of how the mechanism of the pattern is designed by the alternative design is documented).
- Representation of Alternative Design Solution (if exists, class diagram / source code)
- Comparison of Alternative Design Solution to a GoF Design Pattern with respect to quality attributes (YES/NO).

At this point it is necessary to clarify that all selected articles have been examined by all three authors, who have separately assigned values for each variable, for every considered primary study. The final variable values have been assigned to primary studies after discussion on each author's opinion.

2.5 Data Analysis

In a systematic literature review a very important step in order to draw valuable conclusions, is data synthesis. In this step, data from all studies are put together so as to create a data set that can be analyzed in order to answer the research questions. The data synthesis plan in our study aims at accessing data needed for answering each research question, as shown in Table 2.

Table 2. Data Synthesis Overview

Research Question	Data Synthesis
RQ1	Count of discreet alternative design solutions to each GoF design pattern
RQ2	Count of discreet alternative design solutions to each GoF design pattern Count of discreet representation type for alternative design solutions to each GoF pattern
RQ3	Count of discreet alternative design solutions to each GoF design pattern Count of discreet alternative design solutions that have already been compared to each GoF pattern

3. RESULTS

In Table 3, we present the patterns that have been linked to an alternative design solution. More specifically, we mention the patterns for which we have identified alternative design solutions, the studies where such solutions are referenced, the way each study presented the solution and if a comparison of the pattern and the alternative solutions has been performed.

Table 3. Primary Studies Data Set

GoF Design Pattern	Study	Class Diagram	Source Code	Comparison
Bridge	[3]	YES	NO	YES
	[14]	YES	NO	YES
	[16]	YES	YES	NO
	[21]	YES	NO	NO
State	[1]	YES	YES	NO
	[2]	NO	NO	NO
	[3]	YES	NO	YES
	[6]	NO	YES	YES
	[10]	YES	NO	NO
	[24]	NO	NO	NO
	[27]	YES	YES	NO
Abstract Factory	[9]	NO	YES	YES
Mediator	[13]	YES	NO	YES
	[14]	YES	NO	YES
	[30]	YES	YES	YES
Visitor	[12]	YES	YES	NO
	[14]	YES	NO	YES
	[20]	YES	YES	NO
	[23]	YES	YES	NO
	[25]	NO	YES	YES
	[31]	NO	YES	YES
Factory Method	[22]	NO	YES	YES
	[33]	YES	YES	NO
Decorator	[13]	YES	NO	YES
	[22]	YES	YES	YES
	[25]	NO	YES	YES
	[31]	NO	YES	YES
Iterator	[19]	YES	YES	YES
	[26]	YES	YES	NO
Observer	[15]	YES	YES	NO
	[18]	YES	YES	NO
	[25]	NO	YES	YES
	[31]	NO	YES	YES
	[32]	YES	NO	NO
Composite	[25]	NO	YES	YES
	[31]	NO	YES	YES
Proxy	[28]	NO	YES	YES
Strategy	[29]	YES	YES	NO
Command	[8]	YES	YES	NO

Using the data from Table 3 and the data synthesis overview from Table 2, we are able to create a table, with descriptive statistics, that can be used for discussing the research questions of our study.

Table 4. Descriptive Statistics

design pattern	count of alternatives	class diagram	source code	compared with GoF
Bridge	4	4	1	2
State	7	4	3	2
Abstract Factory	1	0	1	1
Mediator	3	3	1	3
Visitor	6	4	5	3
Factory Method	2	1	2	1
Decorator	4	2	3	4
Iterator	2	2	2	1
Observer	5	3	4	2
Composite	2	0	2	2
Proxy	1	0	1	1
Strategy	1	1	1	0
Command	1	1	1	0

4. DISCUSSION

Cataloguing alternative solutions to GoF design patterns is desirable from both a researcher's and a practitioners' point of view. Concerning researchers, a list of alternative designs can aid in the evaluation of design patterns and in the identification of scenarios when the application of a design pattern is desirable. Additionally, we believe that a wide set of proposed designs that solve the same problem can help practitioners to select the most fitting design solutions according to their special needs. In literature, the design alternatives can be introduced as new design patterns or as a piece of code or design artifact, most commonly a class diagram, that solve the same problem as the GoF design pattern. Until now, alternative design solutions have been introduced for at least 13 out of 23 GoF design patterns, which are described in [11], as shown in Figure 1.

The three GoF design patterns that have been linked with most design alternatives are State, Visitor and Observer. State has probably attracted the interest of researchers because it is one of the most widely used patterns. On the other hand, Visitor and Observer have probably been connected to a variety of design alternatives, because they are considered quite complex in their structure and their effect on software quality attributes is controversial [5].

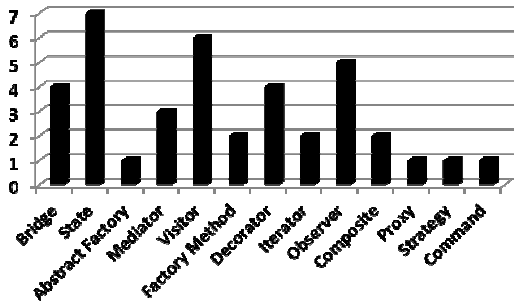

Figure 1. Number of Design Alternatives per GoF Pattern

The alternative designs are divided into two major categories, (a) the ones that have been compared to GoF design patterns and (b) the ones that have been proposed, but no direct comparison to the corresponding GoF pattern has been found.

More specifically, concerning the *Bridge* design pattern, we have identified two alternative design solutions that have been compared to it. In [3], the authors have identified an open-source project that uses cascaded "if" statements in the client class rather than employing the pattern. In [14], the author suggests that using a deeper inheritance tree can substitute the Bridge design pattern. Additionally, [3 and 6] suggest that the *State* pattern can be replaced by a code fragment in the Client class that uses multiple alternative statements to implement the behavior of concrete states and compare their alternatives to the design pattern. Similarly, in [9 and 22], the authors suggest that *Abstract Factory* and *Factory Method* patterns can be implemented with cascaded "if" statements in the Client, reject the polymorphism that the pattern offers and compares the solutions. Furthermore, in [13, 22, 25 and 31], the authors present an alternative to *Decorator*, which uses a deeper inheritance tree to produce an equivalent solution to the design pattern and compare the designs.

Concerning *Mediator*, in [13 and 14], the authors propose and evaluate an alternative solution that does not use the pattern, but offers equivalent functionality, by connecting all related classes directly and not through the Mediator class. In [14, 25 and 31], the authors propose and evaluate an alternative to the *Visitor* pattern. The rationale beneath the alternative solution employs multiple functions rather than "accept" and "visit" methods and only one class hierarchy. The responsibility delegation is handled through a cascaded "if" statement in every subclass of the remaining hierarchy.

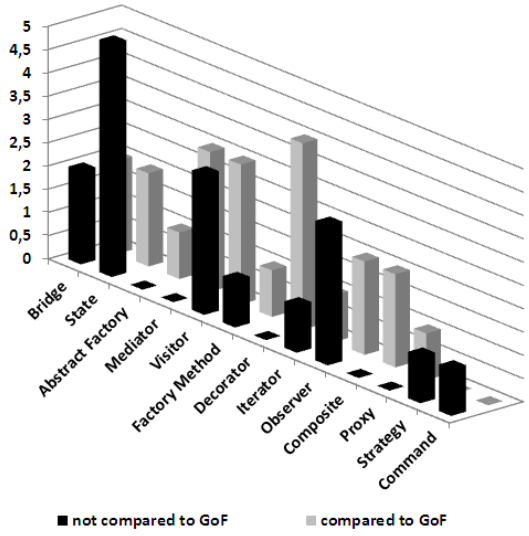

Figure 2. Design Alternatives that are compared to GoF Design Patterns

In [19], the authors present and evaluate three variations of the *Iterator* pattern. The variations are the "Non-Deterministic Iterator", the "Dynamic Iterator" and the "Lazy Recursion Iterator". Concerning *Observer* and *Composite* design patterns, in [25 and 31], the authors omits the alternative solutions in their manuscript, but they are available as supplementary material and a technical report[2]. In both studies, the same set of patterns and alternative solutions are evaluated.

In addition to studies that compare patterns to alternative design solutions, there are several studies, in which new design patterns are proposed and documented. Such research efforts are usually published in pattern specific conferences, such as PLoP and EuroPLoP. At this point it is necessary to clarify that it is out of the scope of this paper to reference all the proposed patterns, but we refer only to those that can be considered alternatives or variations of a GoF design pattern. More specifically, in [1, 2, 10, 24 and 27] the authors present several variations of the State pattern to enhance its behavior. Similarly, in [20 and 23], the authors attempt to classify and organize the variations of the Visitor design pattern. In [12], Gamma introduces a new design pattern, *Extension Object*, which can be used for similar reasons as the Visitor design pattern, in the sense that both patterns enable the extension of object instances. In [21], the author introduces the *Cascading Bridge* design pattern that is a variation of the Bridge GoF pattern. Furthermore, in [16], the author introduces the *TypeObject* that can be considered similar to the Bridge pattern. Additionally, the *Courier* [30] presents a decoupled

[2] http://page.mi.fu-berlin.de/prechelt/packages/tcheck_package.zip
http://page.mi.fu-berlin.de/prechelt/Biblio/wustl_pattern34-1997.pdf

alternative to the Mediator pattern. In [28] two variations of the Proxy pattern are discussed: *Distributed Proxy* and *Remote Proxy*. An extension of the Strategy pattern is presented in [29], for the pattern to handle parameterized algorithms. In [15, 18 and 32], the authors introduce *Middle Observer*, *Dynamic Templat*e, and *Decoupled Reference*, which are a variation of the GoF Observer pattern. In [8], the authors present *"Command Dispatcher"* that is an alternative to the Command design pattern. Moreover, in [26], the authors present a discussion on an alternative for the Iterator design pattern. Finally, [33] introduces alternatives for creating objects, similarly to Factory.

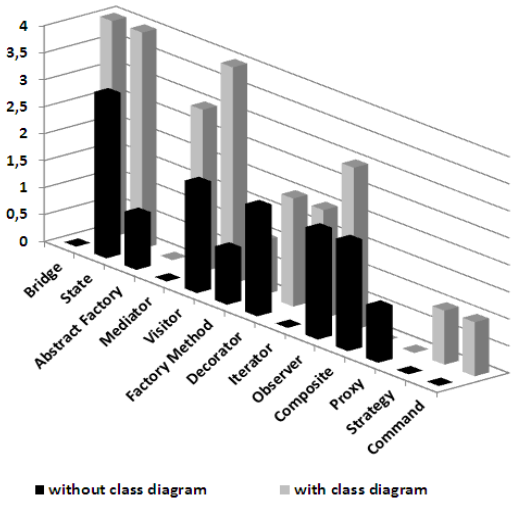

Figure 3. Design Alternatives that are presented through a Class Diagram

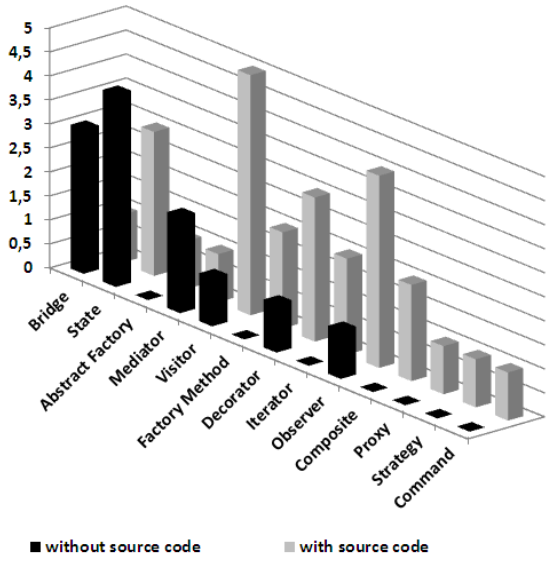

Figure 4. Design Alternatives that are presented through Source Code

Concerning alternative design types of representation, we observe that only 5% of the proposed design alternatives are textually presented, without source code nor class diagram representation. On the other hand, only about 38% of the proposed alternative solutions are fully described, both by providing a sample source code of the alternative and its class diagram. Comparing the two possible ways of representation, we observed that presenting an alternative design solution through its source code is more popular among researchers, although the difference is quite small.

5. CONCLUSIONS

This paper aims at providing a catalogue of design solutions that can be used as alternative solutions, when the use of a GoF design pattern is not the optimum way for designing a requirement. As such, these alternatives are functionally equivalent to design patterns.

In order to achieve this goal, we performed a systematic literature review, from which we identified 39 alternative design solutions for 13 GoF design patterns. Among them, the most alternative design solutions have been proposed for *State*, *Visitor* and *Observer* design pattern. The identified alternative design solutions have been divided into two main categories, solutions that have been structurally compared to design patterns (~55%) and solutions that are only presented as alternatives, but are not evaluated (~45%). In addition to that, 95% of the proposed alternative design solutions are introduced with a formal representation, i.e. class diagram or source code. However, less than 40% of the proposed design alternatives have been introduced with both class diagram and source code.

As future work, we plan to investigate the extent to which design alternatives are used in practice through empirical methodologies, such as case studies and surveys. In addition to that, a more in depth analysis on the methods used for comparing the structural quality of GoF design patterns and alternative design solutions are in progress.

6. REFERENCES

[1] P. Adamczyk, "Selected Patterns for Implementing Finite State Machines", *11th Conference on Pattern Languages of Programs (PLOP '04)*, Monticello, Illinois, 8 – 12 September 2004

[2] P. Adamczyk, "The Anthology of the Finite State Machine Design Patterns", *10th Conference on Pattern Languages of Programs (PLOP '03)*, Monticello, Illinois, 8 – 12 September 2003.

[3] 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.

[4] A. Ampatzoglou, G. Frantzeskou and I. Stamelos, "A Methodology to Assess the Impact of Design Patterns on Software Quality", *Information and Software Technology*, Elsevier, 54 (4), pp. 331-346, April 2012.

[5] A. Ampatzoglou, S. Charalampidou and I. Stamelos, "Research State of the Art on GoF Design Patterns: A Mapping Study", *Journal of Systems and Software*, Elsevier, 86 (7), pp. 1945-1964, July 2013.

[6] B. Baudry, Y. Le Sunye and J. M. Jezequel, "Towards a Safe Use of Design Patterns to Improve OO Software Testability", *Proceedings of the 12th International Symposium on Software*

Reliability Engineering, IEEE, pp. 324, Hong Kong, China, 27-30 November 2001.

[7] K. Y. Cai and D. Card, "An analysis of topics in software engineering - 2006", *Journal of Systems and Software*, Elsevier, 81 (6), pp. 1051 – 1058, June 2008.

[8] B. Dupire and E. B. Fernandez, "The Command Dispatcher Pattern", *8th Conference on Pattern Languages of Programs (PLOP '01)*, Monticello, Illinois, 11-15 September 2001.

[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] L. L. Ferreira and C. M. F. Rubira, "The Reflective State Pattern", *5th Conference on Pattern Languages of Programs (PLOP '98)*, Monticello, Illinois, 11-14 August 1998.

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

[12] E. Gamma, "The Extension Objects Pattern", *3rd Conference on Pattern Languages of Programs (PLOP '96)*, Monticello, Illinois, 4-6 September 1996.

[13] N. L. Hsueh, P. H. Chu and W. Chu, "A quantitative approach for evaluating the quality of design patterns", *Journal of Systems and Software*, Elsevier, 81 (8), pp. 1430-1439, August 2008.

[14] B. Huston, "The effects of design pattern application on metric scores", *Journal of Systems and Software*, Elsevier, 58 (3), pp.261-269, September 2001.

[15] P. Iaría and U. Chesini, "Refining the Observer Pattern: The Middle Observer Pattern", *5th Conference on Pattern Languages of Programs (PLOP '98)*, Monticello, Illinois, 11-14 August 1998.

[16] R. Johnson and B. Woolf, "The Type Object Pattern", *3rd Conference on Pattern Languages of Programs (PLOP '96)*, Monticello, Illinois, 4-6 September 1996.

[17] B. Kitchenham, S. Charters, "Guidelines for performing systematic literature reviews in software engineering", *Technical Report EBSE-2007-001,* Keele University & Durham University Joint Report, Staffordshire, UK, 2007.

[18] F. D. Lyardet, "The Dynamic Template Pattern", *4th Conference on Pattern Languages of Programs (PLOP '97)*, Monticello, Illinois, 3-5 September 1997.

[19] M. L. Nelson, "A Design Pattern for Autonomous Vehicle Software Control Architectures", *23rd International Computer Software and Applications Conference (COMPSAC'99),* IEEE, p.p. 172, Phoenix, Arizona, 25-26 October 1999.

[20] Y. Mai and M. De Champlain, "Pattern Language To Visitors", *8th Conference on Pattern Languages of Programs (PLOP '01)*, Monticello, Illinois, 11-15 September 2001.

[21] B. McCarthy, "The Cascading Bridge Design Pattern", *5th Conference on Pattern Languages of Programs (PLOP '98)*, Monticello, Illinois, 11-14 August 1998.

[22] T. Muraki and M. Saeki, "Metrics for applying GOF design patterns in refactoring processes", *Proceedings of the 4th International Workshop on Principles of Software Evolution,* IEEE, pp.27-36, Vienna, Austria, 10-11 September 2001.

[23] M.E. Nordberg III, "Variations on the Visitor Pattern", *3rd Conference on Pattern Languages of Programs (PLOP '96)*, Monticello, Illinois, 4-6 September 1996.

[24] J. Odrowski and P. Sogaard, "Pattern Integration - Variations of State", *3rd Conference on Pattern Languages of Programs (PLOP '96)*, Monticello, Illinois, 4-6 September 1996.

[25] 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

[26] M. Raner, "The Mutator Pattern", *Proceedings of the 2006 conference on Pattern languages of programs (PLOP '06)*, ACM, Portland, Oregon, 21-23 October 2006.

[27] A. V. Saúde, R. A. S. S. Victório and Gabriel C. A. Coutinho, "Persistent State Pattern", *17th Conference on Pattern Languages of Programs (PLOP '10)*, Reno/Tahoe, Nevada , 16-18 October 2010.

[28] R. Silva, "Distributed Proxy: A Design Pattern for Distributed Object Communication", *4th Conference on Pattern Languages of Programs (PLOP '97)*, Monticello, Illinois, 3-5 September 1997.

[29] O. Sobajic, M. Moussavi and B. Far, "Extending the Strategy Pattern for parameterized Algorithms", *17th Conference on Pattern Languages of Programs (PLOP '10)*, Reno/Tahoe, Nevada, 16-18 October 2010.

[30] R. Switzer, "Courier Patterns", *5th Conference on Pattern Languages of Programs (PLOP '98)*, Monticello, Illinois, 11-14 August 1998.

[31] 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.

[32] P.L. Weibel, "The decoupled reference pattern", *2nd Conference on European Pattern Languages of Programs (EuroPLoP' 96)*, Kloster, Irsee, Germany, 11-13 July 1996.

[33] L. Welicki, J. W. Yoder and R. Wirfs-Brock, "The Dynamic Factory Pattern", *15th Conference on Pattern Languages of Programs (PLOP '08)*, Nashville, Tennessee, 18-20 October 2008.

[34] P. Wendorff, "Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Professional Project", *IEEE Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR 2001)*, pp.77-84, 14–16 March 2001, Lisbon, Portugal.