The Journal of Systems and Software 86 (2013) 1945–1964

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Research state of the art on GoF design patterns: A mapping study

Apostolos Ampatzoglou[a,*], Sofia Charalampidou[b], Ioannis Stamelos[a]

[a] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece
[b] Department of Computer Science & Technology, Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

*Article history:*
Received 27 October 2011
Received in revised form 12 March 2013
Accepted 14 March 2013
Available online 26 March 2013

*Keywords:*
Design patterns
Mapping study
Software quality attributes

ABSTRACT

Design patterns are used in software development to provide reusable and documented solutions to common design problems. Although many studies have explored various aspects of design patterns, no research summarizing the state of research related to design patterns existed up to now. This paper presents the results of a mapping study of about 120 primary studies, to provide an overview of the research efforts on Gang of Four (GoF) design patterns. The research questions of this study deal with (a) if design pattern research can be further categorized in research subtopics, (b) which of the above subtopics are the most active ones and (c) what is the reported effect of GoF patterns on software quality attributes. The results suggest that design pattern research can be further categorized to research on GoF patterns formalization, detection and application and on the effect of GoF patterns on software quality attributes. Concerning the intensity of research activity of the abovementioned subtopics, research on pattern detection and on the effect of GoF patterns on software quality attributes appear to be the most active ones. Finally, the reported research to date on the effect of GoF patterns on software quality attributes are controversial; because some studies identify one pattern's effect as beneficial whereas others report the same pattern's effect as harmful.

## 1. Introduction

Patterns have been introduced by Christopher Alexander in the field of architecture, where he documented reusable architectural proposals for producing good quality designs (Alexander et al., 1977). In the mid-90s, the idea of patterns was adopted by object-oriented software developers. In Gamma et al. (1995), the so-called GoF (Gang of Four, Gamma, Helms, Johnson and Vlisides), cataloged 23 design patterns aimed at meeting some commonly-recurring object-oriented design needs.

In recent years, GoF design patterns have attracted the attention of researchers and they are now considered a respectable part of software engineering research and practice. However, until now, there has not been any published review that summarizes the research state of the art in this area. Even though we focus on GoF patterns, GoF patterns are not the only software patterns. For example, other pattern catalogs introduce architectural patterns (Avgeriou and Zdun, 2005; Buschmann et al., 1996), i.e., design patterns on a higher-level of abstraction than objects (component/subsystem interactions), game design patterns (Bjork et al., 2003), i.e., patterns on game logic, etc. This study focuses on GoF patterns although it does not suggest that GoF patterns are *better practices* than other design patterns. However, the interest that GoF patterns have attracted from both academia and industry justifies the scope of this study. Furthermore, intheir mapping study Zhang and Budgen (2012) note that all their available primary studies deal with GoF patterns, which therefore supports our contention that GoF patterns form a reasonable object of study. Additionally, it is expected that if we widen the scope of our study to include other types of patterns, there will be no real obvious boundary.

The goal of this paper is to summarize the existing research work on *GoF design patterns*, later referenced as *design patterns*, through a mapping study, which is sometimes considered as a form of a systematic literature review. Recently, the number of systematic literature reviews has increased in Software Engineering. At this point, it is estimated that over 150 SLRs have been published (da Silva et al., 2010; Kitchenham et al., 2009, 2011; Kitchenham et al., 2010). In Section 2, we discuss related work and in Section 3 we provide background information on software quality attributes. In Section 4, we present an overview of the followed methodology and define the research questions that our study will investigate. Section 5 presents an overview of the primary studies and accumulated data from the primary studies. The discussion of the paper (see Section 6) is divided in subsections, according to the research goals, i.e., *active research subtopics and effect on quality attributes.* Finally, threats to validity and conclusions are described in Sections 7 and 8, respectively.

* Corresponding author. Tel.: +30 2310348007.
*E-mail addresses:* apamp@csd.auth.gr (A. Ampatzoglou), stamelos@csd.auth.gr (I. Stamelos).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.03.063

## 2. Related work

In this section, we present some previous studies which either perform a systematic review of pattern literature or catalog the effect of patterns on software quality attributes. Under this perspective, we identified three studies that are close to our work, which however have important differences.

In (Galster and Avgeriou, 2012), Galster and Avgeriou summarize the effect of software architecture patterns (SOA patterns) on quality attributes. The authors have related more than 70 SOA patterns, with quality attributes based on the pattern description on the catalog that they have been introduced. Their results indicate a mismatch between patterns for service-based systems and quality attributes that are considered important for service-based systems. The main difference of this work, with respect to our study, is the focus on SOA patterns rather than GoF patterns.

A similar work on GoF patterns has been performed by Khomh and Gueheneuc [P62], where the authors evaluated the effect of all GoF patterns on software quality attributes through a survey. The results suggested that, in constrast to common beliefs, design patterns in practice impact negatively several quality attributes. The difference of [P62] in comparison to this study is the use of a different research method. [P62] is a survey whereas this paper is a mapping study. Thus, [P62] is one of the 33 studies that have been taken into account while investigating the effect of GoF design patterns on software quality attributes.

In Zhang and Budgen (2012), Zhang and Budgen performed a systematic literature review on the effectiveness of software design patterns, on articles published until 2009. More specifically, the research method used was a systematic literature review on empirical studies concerning design patterns and software quality attributes. The main difference of Zhang and Budgen (2012) with respect to this paper is the aim of the two literature reviews. In our study, we do not aim only on summarizing empirical evidence, but to gather a broader dataset, concerning GoF design pattern research. In addition to that, we do not only focus on the effect of patterns on quality attributes, but introduce GoF design patterns research subtopics, as well.

## 3. Software quality attributes

Software quality models are usually hierarchical (Dormey, 1995; ISO9126, 1992). In this paper, we use ISO/IEC 9126 as reference model for discussing the effect of design patterns on software quality (ISO9126, 1992). The first level of ISO 9126 describes six quality attributes, i.e., portability, functionality, reliability, usability, efficiency, and maintainability, which are further divided in several sub-characteristics as shown in Fig. 1. Next, each quality sub-attribute (low-level quality attributes, such as complexity, cohesion, etc.), can be assessed by a set of metrics that can be used as indicators for the score of a system with respect to the corresponding high level quality attribute.

One of the major concerns of a developer who employs a pattern is the quality attributes of the design, after pattern application. When using patterns, some of the software quality attributes that will be affected are maintainability (Vokáč et al., 2004), understandability [P40] and reliability [P62]. However, assessing the effect of patterns on software quality is an extremely difficult task. Until now, researchers have attempted to evaluate the use of patterns with empirical, i.e. surveys, case studies, experiments and analytical methods. In most real systems, patterns interact (pattern coupling) and such interactions make the evaluation of the effect of patterns on quality attributes even more difficult.

## 4. Mapping study methodology

To perform our mapping study, we used a well-known methodology for systematic literature reviews (Bereton et al., 2007; Petticrew and Roberts, 2006). The selected methodology is considered a standard for conducting and presenting systematic reviews in software engineering and it is applied in a wide variety of papers (Ampatzoglou and Stamelos, 2010; Cai and Card, 2008; Dyba and Dingsoyr, 2008; Hauge et al., 2010; Heckman and Williams, 2011; Kitchenham et al., 2009; Walia and Carver, 2009). Following (Kitchenham and Charters, 2007; Kitchenham et al., 2009), the mapping study plan consists of six definitions:

(a) Research questions definition
(b) Search process definition
(c) Inclusion and exclusion criteria definition
(d) Quality assessment definition
(e) Data collection process definition
(f) Data analysis definition.

A flow chart that summarizes the review process is presented in Fig. 2. The notions used in Fig. 2, are exactly the ones used in a flow chart, i.e. the nodes represent actions (*Gather Data for Each Study*, *Identify Research Topics*, etc.) and edges represent the transition from one node to another (Zhang and Budgen, 2012).

### 4.1. Research questions

In this study, we planed to investigate several issues concerning the research state-of-the-art on patterns. The research questions have been identified using a Goal-Question-Metrics (GQM) approach (Basili et al., 1994). GQM defines a top down approach that

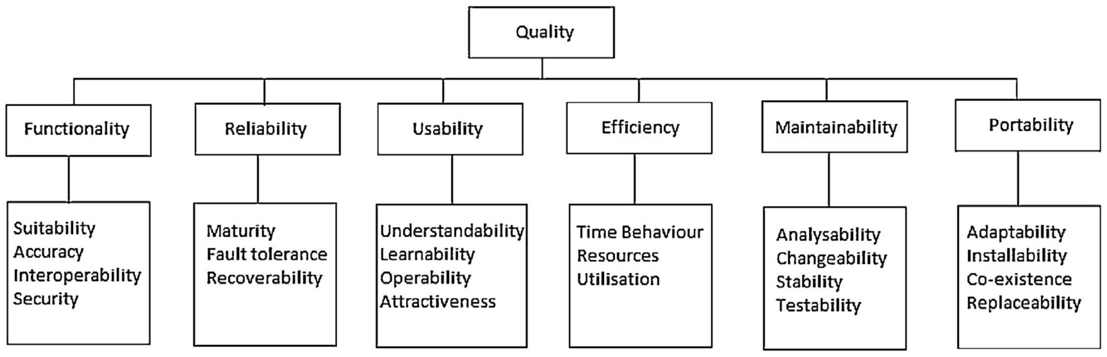

**Fig. 1.** ISO 9126 quality attributes.

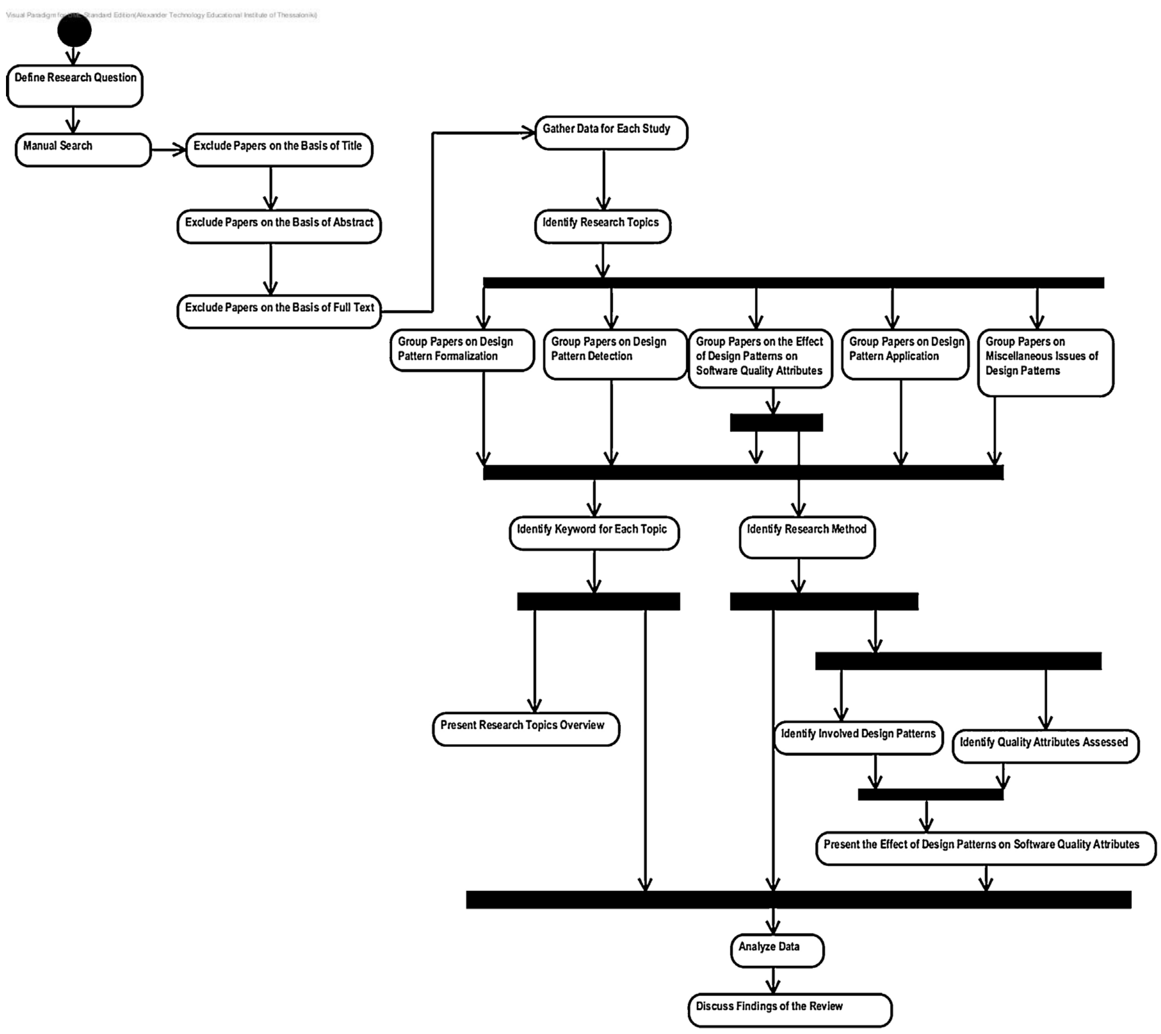
Define Research Question
Manual Search
Exclude Papers on the Basis of Title
Exclude Papers on the Basis of Abstract
Exclude Papers on the Basis of Full Text
Gather Data for Each Study
Identify Research Topics
Group Papers on Design Pattern Formalization
Group Papers on Design Pattern Detection
Group Papers on the Effect of Design Patterns on Software Quality Attributes
Group Papers on Design Pattern Application
Group Papers on Miscellaneous Issues of Design Patterns
Identify Keyword for Each Topic
Identify Research Method
Present Research Topics Overview
Identify Involved Design Patterns
Identify Quality Attributes Assessed
Present the Effect of Design Patterns on Software Quality Attributes
Analyze Data
Discuss Findings of the Review

**Fig. 2.** Mapping study process–flow diagram.

aims at developing meaningful metrics. The approach introduces three levels:

(a) Conceptual level (goal): "*A goal is defined for an object for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment*" (Basili et al., 1994).
(b) Operational level (question): "*A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal*" (Basili et al., 1994).
(c) Quantitative level (metric): "*A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way*" (Basili et al., 1994).

The goal of the study is to provide researchers with a catalog of related work and possible interesting research areas. In addition to that, concerning practitioners the study aims to provide a quick reference to quality attributes that are related to GoF design patterns. The questions are defined in this section and the metrics are defined in Section 4.6. Concerning the first goal of the paper, i.e., *active research subtopics*, two research questions have been stated, whereas concerning the second goal, i.e., *effect of patterns on quality attributes*, an additional research question has derived.

**RQ$_1$** : *Can design pattern research be further categorized according to more specialized research subtopics?*

This research question is important for providing researchers an overview of trends and gaps in current design pattern research.

**RQ$_2$** : *Which are the most active research subtopics concerning design patterns?*

This research question is important to researchers. The findings related to this question can be used by researchers as a catalog of related work, divided by a rationale categorization. Additionally, the results point to interesting research subtopics and to research subtopics that are neglected up to now.

**RQ3** : *What pattern effects on quality attributes have been identified to date?*

This research question is interesting for both researchers and practitioners. Researchers, who are specialized on the effect of GoF patterns on software quality attributes, can easily identify the controversial reported results on the subject and further investigate them. Practitioners can consult the findings of our study when using a design pattern and take into account their personal needs with respect to relevant software quality attributes.

The names of the quality attributes that are investigated come from ISO 9126 model. However, in several primary studies, we have identified several quality attributes that are not presented in Fig. 1.

**Table 1**
Quality attribute mapping.

| Primary study quality attribute | ISO-based quality attribute | Studies |
|---|---|---|
| Change Proneness | Stability | P44 |
| Reusability | Adaptability | P62, P93 |
| Flexibility | Maintainability | P72 |
| Modularity | Maintainability | P62, P93 |
| Generality | Adaptability | P62 |
| Scalability | Maintainability | P62 |
| Robustness | Reliability | P62 |

The mapping between the terms of primary studies and ISO-based terms are presented in Table 1.

For example [P44] references the *change proneness* quality attribute which is the probability of a class to change. In ISO 9126 *change proneness* is not defined as a quality attribute. The closest ISO-based attribute is *stability* which is the opposite of *change proneness*, i.e., the probability of a class not to change. Thus, *change proneness* is mapped to *stability*, taking into account the negative relationships between quality attributes while evaluating patterns. Finally, some quality attributes of the ISO first level, e.g., usability, are quality in use attributes, whereas patterns are expected to affect internal and external ones[1]. For example, usability is examined from a developer's perspective, i.e. how understandable and attractive the code that uses a pattern is.

## 4.2. *Search process*

The search process of our research has been based on the process described in (Cai and Card, 2008), where the authors selected seven journals and seven conferences as search space. The journals have been selected according to their impact factor (greater than 0.800), whereas the conferences have been selected according to their acceptance rate (about 30%). After creating the selected venue list, we observed that our study explores the journals and conferences of Cai and Card (2008) and investigated four additional journals, six additional conferences, and two additional workshops that deal with reverse engineering, maintenance, refactoring, metrics, and generic software engineering. The journals and conferences that have been explored are presented in Table 2, along with their impact factor or acceptance rate.[2] The topic of the journals and conferences must strictly be software engineering. Thus, venues such as *IEEE Computer*, *Journal of the ACM*, or *Communication of the ACM*, although of very high quality and impact factor, were excluded. Software architecture conferences have not been considered in the search space because we assumed that the majority of papers published in such venues would deal with architectural patterns and not GoF design patterns. Concerning the time period of the search process, the study has not defined any starting search date and includes articles published until the end of 2010, i.e., all editions of selected conference and all volumes of selected journals were considered in the review process. The search process was conducted by a manual search through the portals of five digital libraries, namely ACM, IEEE, ScienceDirect, Springer, and Wiley. The mapping among venues and digital libraries is presented in Table 2. The only term used in the search process was *pattern*, referenced in the title of the publication. The exclusion of non-relevant articles was conducted manually according to the article filtering criteria defined in Section 4.3.

In Zhang and Budgen (2012) it is suggested that many papers might include one or more studies and that one study might be reported by one or more papers. In the field of software engineering, a common practice among researchers is to publish their early research results in conference proceedings in order to get quicker feedback from the research community, as a means for evolving and maturing their work. In most of the cases the final outcome of a study is a publication to a software engineering journal. In this work we have grouped papers into studies and report them both. The main criterion for merging papers into studies was the similarity of research method and questions.[3] The papers that have been merged into one study are presented in Appendix C.

## 4.3. *Article filtering phases*

The papers that are selected as primary studies in the review must be relevant to an object-oriented design pattern described in (Gamma et al., 1995). In line with (Dyba and Dingsoyr, 2008), there are four stages of filtering the article set to produce the primary study data set. These stages are presented in Table 3. In line with (Bereton et al., 2007), the search process was handled by one of the three authors.

The article search process returned a set of studies that included all publications that used the term pattern in their title, without evaluating its relevance to GoF design patterns. On the completion of this phase the article set (413 articles), went through a manual inspection of their titles. Although the search process returned a set of primary studies that included the term pattern in the title, there was no article exclusion regarding its relevance to a GoF design pattern at that stage. For example, a paper entitled *Performance of circuit-switched interconnection networks under no uniform traffic patterns*, was excluded in the second phase because the paper is clearly unrelated to GoF design patterns.

On the completion of the abovementioned phase, the candidate primary study set included 215 papers. In the next phase, one author assessed the relevance of each paper by examining its abstract. The most common exclusion criterion at this phase proved to be the relation of the candidate primary study to architectural patterns or human computer interaction patterns. Next, the full manuscripts of the 158 articles remaining were examined by all three authors independently. In this phase, any article that had no strict reference to at least one GoF design pattern was excluded. No conflicts among the authors' opinions arose. Finally, all 118 articles that successfully passed all previously mentioned phases have been included in the review without applying further criteria. The inclusion/exclusion criteria are explicitly listed below:

- Inclusion criteria:
  - papers dealing with software design patterns
  - explicit reference to one GoF design pattern
- Exclusion criteria:
  - literature that was only available in the form of abstract
  - literature in the form of a poster or a short paper (less than 5 pages)

## 4.4. *Quality assessment*

The quality of a systematic review is highly correlated to the quality of the primary studies in the sense that the results and the

[1] Quality in use: Type of quality that is perceived when the final product is used in real conditions. Internal Quality: Type of quality that is perceived from a non-executable view of the software (static). External Quality: Type of that is perceived from running software (Kitchenham and Pfleeger, 1996).

[2] The conference acceptance rates are from (http://people.engr.ncsu.edu/txie/seconferences.htm) for the year 2010. If 2010 is not available, we take into account the last known acceptance rate. The journals' impact factors have been extracted from (http://www.isiwebofknowledge.com).

[3] The complete list of comparisons among papers, so as to merged them into studies is provided in the web, see http://students.csd.auth.gr/~apamp/mapping_study_all_authors.xlsx.

**Table 2**
Publication venues.

| Name | # papers | Impact factor/acceptance rate | Digital sources |
|---|---|---|---|
| Annual Computer Software and Application Conference (COMPSAC) | 13 | 31% | IEEE |
| European Conference on Software Maintenance and Reengineering (CSMR) | 13 | 30% | IEEE |
| International Conference on Software Engineering (ICSE) | 10 | 14% | IEEE |
| International Conference on Software Maintenance (ICSM) | 10 | 26% | IEEE |
| ICSE Workshops | 9 | N/A | IEEE |
| IEEE Working Conference on Reverse Engineering (WCRE) | 9 | 25% | IEEE |
| IEEE Transactions on Software Engineering (TSE) | 8 | 2.265 | IEEE |
| Journal of Systems and Software (JSS) | 7 | 1.293 | Science Direct |
| Information and Software Technology (IST) | 6 | 1.527 | Science Direct |
| International Conference on Automated Software Engineering (ASE) | 6 | 18% | IEEE/ACM |
| Object Oriented Programming, Systems, Languages and Applications (OOPSLA) | 6 | 28% | ACM |
| International Conference on Program Comprehension (ICPC) | 4 | 27% | IEEE |
| IEEE Metrics Symposium (METRICS) | 3 | 29% | IEEE |
| Symposium on Empirical Software Engineering and Measurement (ESEM) | 3 | 29% | IEEE/ACM |
| IEEE Software (IEEESoft) | 2 | 1.511 | IEEE |
| Empirical Software Engineering (ESE) | 2 | 1.796 | Springer |
| International Symposium on Software Reliability Engineering (ISSRE) | 2 | 25% | IEEE |
| ACM SIGSOFT Symposium on Foundation of Software Engineering (FSE) | 1 | 20% | ACM |
| Advancements in Software Engineering (AdSE) | 1 | 1.004 | Elsevier |
| ACM Transactions on Programming Languages and Systems (TOPLAS) | 1 | 1.167 | ACM |
| FSE Workshops | 1 | N/A | ACM |
| Journal of Software: Evolution and Process | 1 | 0.844 | Wiley |
| Software Testing, Verification and Reliability (STVR) | 1 | 0.957 | Wiley |
| ACM Transactions on Software Engineering and Methodology (TOSEM) | 0 | 1.694 | ACM |
| Automated Software Engineering Journal (ASEJ) | 0 | 0.806 | Springer |
| International Symposium in Software Testing and Analysis (ISSTA) | 0 | 23% | ACM |
| Requirements Engineering Journal (RE) | 0 | 0.862 | Springer |
| Science of Computer Programming (SCP) | 0 | 1.306 | Science Direct |
| Software and Systems Modeling (SoSyM) | 0 | 1.404 | Springer |

conclusions of the secondary study are based on the findings of the primary studies. Thus, in a review, it is crucial to include primary studies that are methodologically sound and that are clearly presenting their results. In our review, we have included papers that are published in top journals and conferences, and workshops that are held in conjunction with top conferences in software engineering. Conferences and workshops have been included in the search space, because many good software engineering papers are presented at international conferences or workshops that usually cover more current and up-to-date research advancements; because their review and publication period is shorter than that for journals, without significant trade-off concerning the quality of the articles. Hence, although we have not performed a systematic quality assessment for the primary studies in our review, we believe that the quality of the selected papers is good enough for the purpose of our study.

### 4.5. Data collection

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

[$A_1$] Type of publication (journal, conference, workshop)
[$A_2$] Published in (journal or conference name)
[$A_3$] Year of publication
[$A_4$] Keywords (the keywords have been extracted from authors' expert judgment)

**Table 3**
Article inclusion–exclusion phases.

| Step | Remaining papers |
|---|---|
| Identify relevant studies–search digital libraries | 413 |
| Exclude studies on the basis of titles | 215 |
| Exclude studies on the basis of abstracts | 158 |
| Obtain studies and select the most relevant to design patterns on the basis of full text | 118 |

For the studies that deal with software quality, additional information has been retrieved:

[$Q_1$] Patterns investigated (name of pattern)
[$Q_2$] Quality attributes investigated
[$Q_3$] Software metrics used (if any)
[$Q_4$] Research method used

All retained articles have been examined by all three authors. Every author has completed data extraction for every primary study separately. Then the values of each variable (obtained by each author) have been compared to each other and its final value has been assigned to the primary study after discussion on every author's opinion. If two or more authors assigned the same value to one variable this value was assigned to the variable without further discussion. In any other case after a debate among the authors a value was assigned to every variable. In total, 72 conflicts have been resolved concerning 44 primary studies.

### 4.6. Data analysis

The data collected for variables, *type of publication* ($A_1$), *published in* ($A_2$), *year of publication* ($A_3$), and *research method used* ($Q_4$) were used to provide descriptive statistics, mostly frequency tables, on design pattern research. The *keywords* ($A_4$) variables were used to identify and discuss the most active research subtopics that deal with design patterns and to aid in the description of research state of the art on every subtopic (*addressing* $RQ_1$ and $RQ_2$). Concerning the discussion on the effect of each pattern on software quality attributes (*addressing* $RQ_3$), variables *pattern investigated* ($Q_1$), *quality attributes investigated* ($Q_2$), and *software metrics used* ($Q_3$) were taken into account.

In a mapping study an important step toward drawing valuable conclusions is the categorization of the papers that have been found. In this step, data from all studies are put together so as to create a data set that can be analyzed to answer the research questions. The data categorization plan in our study aims at accessing data needed for answering every research question from many perspectives, as shown in Table 4. The topics emerged during analysis, by

**Table 4**
Data categorization overview.

| RQ | Basic measures | Variables used |
|---|---|---|
| *RQ*$_1$ | Count of keywords<br>Count of articles per year for every research subtopic<br>Count of keywords per research subtopic | Keywords ($A_4$)<br>Year of publication ($A_3$)<br>Type of publication ($A_1$)<br>Published in ($A_2$) |
| *RQ*$_2$ | Count of research methods used for investigating the effect of GoF patterns on software quality attributes | Keywords ($A_4$)<br>Research method used ($Q_4$) |
| *RQ*$_3$ | Mapping among design patterns & software quality attributes<br>Count of positive and negative critiques on every pattern–quality attribute pair | Pattern investigated ($Q_1$)<br>Quality attributes investigated ($Q_2$)<br>Software metrics used ($Q_3$) |

**Table 5**
Primary study keywords frequency.

| Keyword | # papers | # studies | Brief description |
|---|---|---|---|
| Detection | 36 | 30 | Papers that present algorithms, tools or methods that can be used to identify GoF design pattern instances in source or binary code. |
| Specification | 11 | 10 | Papers that discuss possible ways of presenting and specifying GoF design patterns. |
| Detection Algorithm | 10 | 8 | Papers that present algorithms that can be used to identify GoF design pattern instances in source or binary code. |
| Quality | 10 | 10 | Papers that deal with the effect of GoF design patterns on software quality. |
| Detection Accuracy | 9 | 9 | Papers that deal with the detection accuracy of algorithms, tools or methods that can be used to identify GoF design pattern instances in source or binary code. |
| Detection Tool | 9 | 9 | Papers that present tools that can be used to identify GoF design pattern instances in source or binary code. |
| Detection Technique | 8 | 8 | Papers that present techniques that can be used to identify GoF design pattern instances in source or binary code. |
| Implementation | 7 | 6 | Papers that exhibit how GoF design patterns can be implemented on various languages, or automatically applied. |
| Maintainability | 7 | 7 | Papers that deal with the effect of GoF design patterns on software maintenace. |
| Structural | 7 | 3 | Papers that somehow deal with only one category of GoF design patterns, i.e. structural patterns |
| Visualization | 6 | 5 | Papers that present possible ways of visualizing GoF design patterns. |

using reciprocal translation, which is a content analysis approach (Noblit and Hare, 1988). The process of reciprocal translation is used to express the concept of each study, in relation to the concept of other studies. An alternative approach for classifying research on patterns is the ACM Classification Schema (Cai and Card, 2008). Having used this approach, most studies would have primarily been classified in D.2.10 (Design). Secondary categorizations would classify the studies to D.2.2 (Design Tool) for design pattern detection, D.2.3 (Coding Techniques) for pattern application, D.2.7 (Maintenance) for the effect of patterns on maintainability, D.2.8 (Metrics) for the effect of patterns on metric values and D.2.13 (Reusable Software) for the effect of patterns on reusability.

## 5. Results

This section of the paper presents the results of this mapping study, according to the three research questons. The first section (Section 5.1) presents the research subtopics that have been identified according to our analysis. Next, an overview of the primary studies included in the review, handling each study separately is provided (Section 5.2). The third part (Section 5.3) provides descriptive statistics, which derived from synthesizing the results of the analysis, which will be used for discussing the effect of patterns on software quality. The complete dataset of each study is available in the web.[4]

### 5.1. Research subtopics & top publishers identification

The first research question of our study aims at the identification of specialized research subtopics that can further categorize research on GoF design patterns. To achieve this goal, we qualitatively analyzed and synthesized the most common keywords that have been extracted during the review process. An issue with qualitative analysis of this form is that it really needs to be performed systematically. In order to eliminate posibble inconsistencies among terms used in different studies we used reciprocal translation for the brief descriptions provided for each keyword. The most common keywords that have been identified from all primary studies are presented in Table 5, accompanied by their brief descriptions.

The analysis has been performed on the brief descriptions of the keywords, where we attempted to identify similarities among them. Keywords with similar brief descriptions have been put together. For example, keywords *Detection, Detection Algorithm, Detection Tool, Detection Tool Accuracy* and *Detection Technique* all share the common aim of identifying GoF design pattern instances from existing projects. This aim was not a part, of any other brief description. Thus, *Design Pattern Detection* is identified as a GoF Design Pattern research subtopic.

After performing the abovementioned process for all keywords, the research state-of-the-art on design patterns has been divided into five research subtopics, as they have been identified by our data analysis, as follows:

- *Design Pattern Formalization* (from keywords Specification and Visualization), deals with papers that attempt to create ontologies, markup languages, and so on, to describe design patterns.
- *Design Patterns and Software Quality subtopic* (from keywords Quality and Maintainability), deals with papers that investigate the effect of design pattern application on software quality.
- *Design Pattern Detection* (from keywords Detection, Detection Algorithm, Detection Tool, Detection Tool Accuracy and Detection Technique), includes papers that deal with methodologies,

[4] http://students.csd.auth.gr/~apamp/mapping_study.html.

**Table 6**
Top Publishers of papers on design patterns.

| Name | # papers | # studies | Papers |
|---|---|---|---|
| Y. G. Guéhéneuc | 11 | 10 | P2, P33, P45, P46, P57, P58, P59, P62, P63, P66, P90 |
| C. Gravino | 7 | 4 | P27, P28, P29, P30,P31, P32, P98 |
| M. Risi | 7 | 4 | P27, P28, P29, P30,P31, P32, P98 |
| A. De Lucia | 6 | 3 | P27, P28, P29, P30,P31, P32 |
| V. Deufemia | 6 | 3 | P27, P28, P29, P30,P31, P32 |
| G. Antoniol | 6 | 4 | P4, P5, P6, P33, P45 |
| A. Chatzigeorgiou | 4 | 4 | P3, P66, P67, P109 |
| T. H. Ng | 4 | 4 | P78, P79, P80, P81 |
| S. C. Cheung | 4 | 4 | P78, P79, P80, P81 |

algorithms, and tools that mine pattern instances from source code and other artifacts.

- *Design Pattern Application* (from keyword Implementation) subtopic involves papers that present methods for identifying systems that need pattern application or methods and tools that automate or assist the application of patterns.
- *Miscellaneous Issues on Design Patterns*, consists of studies that cannot be classified into any other previous subtopic.

At this point it is necessary to clarify that, some papers are linked to keywords of two subtopics. In such cases, the article has been categorized under only one subtopic, according to authors' judgment. We believe that our classification schema adequately demarcates research subtopics within the overall design pattern research state of the art, in a more balanced way than a generic schema, e.g., ACM Classification Schema, because it is more specialized and focuses on pattern related characteristics that could not have been retrieved by more generic schemas. In the remaining of the paper, we will refer to the identified GoF patterns research subtopics as research subtopics.

A common practice when performing a mapping study is the identification and ranking of the researchers in the domain under study. Table 6, summarizes the publishers with the maximum number of publications in research on the field of design patterns.

## 5.2. Research subtopics intensity and overview

Figs. 3–8 depict the trend of research activity for each subtopic over the time period covered by the mapping study. In addition to that, Table 7 presents the count of published articles within each research subtopic and pointers to the corresponding publications.

The figures suggest that considering the overall research intensity over time, the volume of research is increasing over the years. However, there are years when the number of published papers on design patterns has decreased with respect to the previous year. Concerning research subtopics, research on pattern application was the most intense research subtopic until 2002 (30.23% of overall number of papers), but since then the research effort on this subtopic has faded out. Research on pattern detection and pattern formalization was limited to a small number of papers until 2005 and 2003 respectively. Then, both subtopics were subject to more publications. However, research on pattern detection had a more stable rate of published studies. Additionally, research on the effect

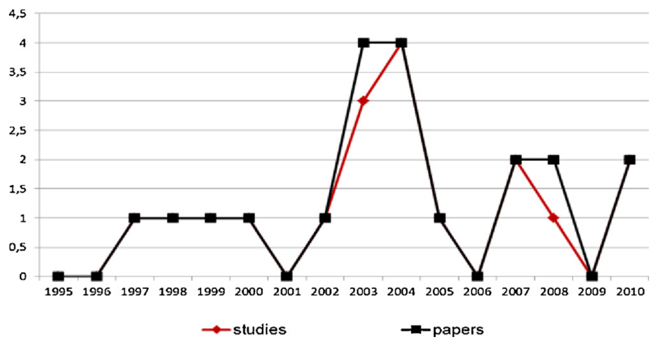

**Fig. 4.** Research intensity over time (design pattern formalization).

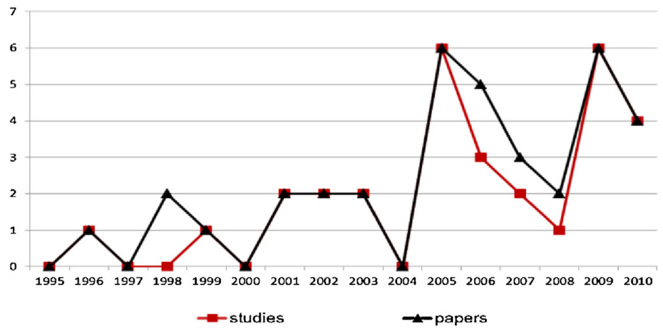

**Fig. 5.** Research intensity over time (design pattern detection).

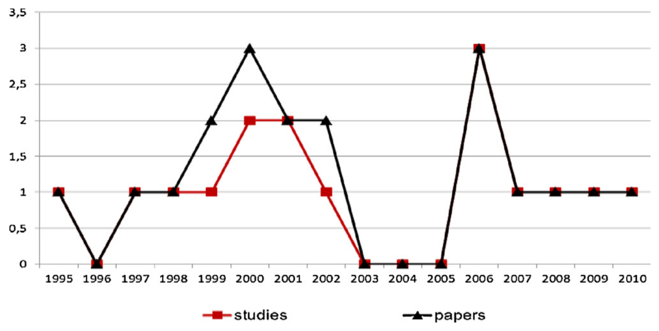

**Fig. 6.** Research intensity over time (design pattern application).

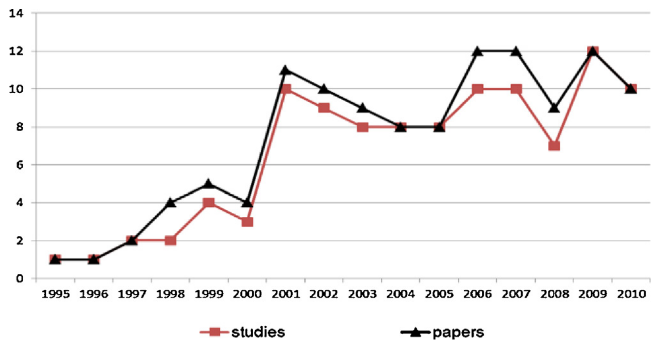

**Fig. 3.** Total research intensity over time.

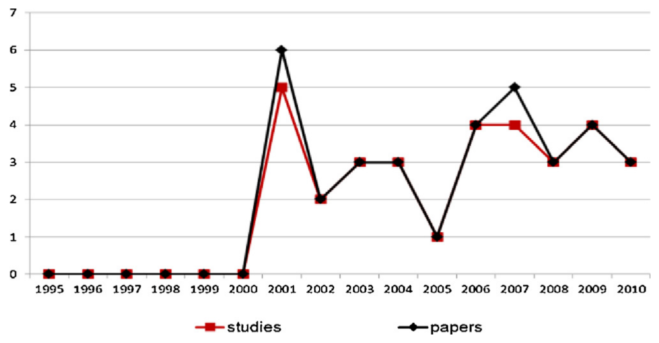

**Fig. 7.** Research intensity over time (effect of design patterns on quality).

**Table 7**
Research subtopics.

| Name | # papers | # studies | Papers |
|---|---|---|---|
| Design Pattern Formalization | 20 (16.95%) | 18 (17.14%) | P1, P11, P14, P15, P18, P19, P34, P35, P37, P38, P42, P64, P71, P74, P76, P85, P90, P101, P104, P118 |
| Design Pattern Detection | 36 (30.51%) | 30 (28.57%) | P2, P4, P5, P6, P8, P10, P23, P27, P28, P29, P30,P31, P32, P36, P41, P43, P45, P51, P54, P58, P59, P61, P65, P66, P68, P83, P88, P89, P96, P100, P102, P109, P112, P113, P114, P117 |
| Design Patterns and Software Quality | 35 (29.66%) | 33 (31.42%) | P3, P9, P12, P13, P16, P33, P39, P40, P44, P46, P47, P48, P49, P52, P55, P56, P57, P62, P63, P67, P72, P73, P77, P78, P79, P80, P81, P91, P92, P93, P98, P99, P110, P111, P115 |
| Design Pattern Application | 18 (15.25%) | 15 (14.28%) | P17, P20, P22, P24, P25, P26, P50, P53, P60, P69, P70,P75, P82, P86, P94, P105, P106, P116 |
| Miscellaneous Issues | 9 (7.63%) | 9 (8.57%) | P7, P21, P84, P87, P95, P97, P103, P107, P108 |

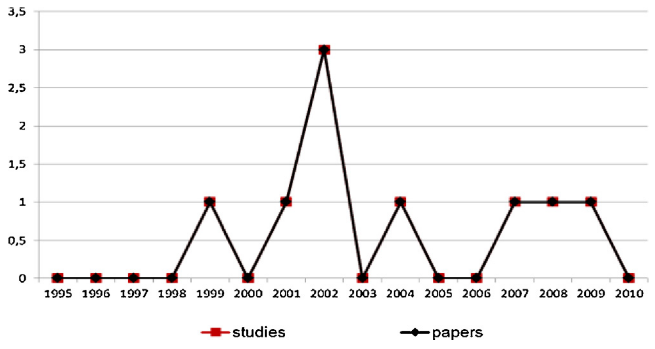

**Fig. 8.** Research intensity over time (design pattern miscellaneous issues).

of patterns on software quality attributes is the most stable research subtopic, because from 2001, there was only one year with fewer than two publications.

Concerning the differences between number of studies and number of papers, we observe that the topic with most studies that have been published in more than one venue is design pattern detection. This is a reasonable result in the sense that design pattern detection algorithms, tools and techniques can more easily evolve than any other kind of research activity. Changing one step of an algorithm, to make it work faster or more accurately, is a result that worths publishing without changing the complete methodology. This is not applicable to other research topics, e.g. the effect of GoF patterns on software quality, where minor changes do not produce publishable results.

Finally, one role for a mapping study is to establish if there are enough primary studies in an area to justify conducting a systematic literature review. In the case of GoF design patterns research we believe that there is room for a systematic literature review in the fields of design pattern detection and on the effect of GoF patterns on software quality attributes.

Next, we provide a detailed description of the state of the art on each subtopic, according to the categories that have been defined in Section 5.1. The papers have been classified in the five subtopics and then grouped in a content-based manner. For each group, a cumulative overview of their goals is provided. A more detailed presentation of each paper is provided in the web.[5] For every research subtopic, we created a list of the most frequent keywords within this subtopic. The keywords can be the method used for the purpose of the study, e.g. *meta-programming languages*, or the actual aim of the study, e.g., *visualization of patterns*. Every keyword is accompanied by two characteristics, i.e., a list of pointers to the articles that it has been identified and its identification frequency within the subtopic. The keywords that are presented in the tables aim at providing indications on research intensity around these keywords and not to clearly describe the primary study.

### 5.2.1. *Design patterns formalization*

The articles that deal with pattern formalizations all share the common aim of investigating, identifying, and specifying innovative approaches that deal with modeling and formalizing patterns. In [P35, P42, and P64] the authors deal with the visualization of design patterns with UML artifacts. These studies provide a way to use design patterns with several practical benefits concerning tools that help practitioners in applying design patterns. In [P19 and P85], the authors present a repository including formal specifications of the problems that each pattern solves and demonstrate a language for formally describing the Visitor pattern to capture its essence in a reusable library. In [P14, P15, and P104], the authors use first order predicate logic to specify the behavioral and structural characteristics of design patterns.

In [P1 and P34], a component-based specification of design patterns is suggested, guided by the design artifacts that are involved in patterns and propose new symbols on class and collaboration diagrams that help in pattern representation. In [P11 and P18], the authors provide tools that are based on constraints and logical graphs to formalize design patterns. [P37, P74 and P101] introduce meta-programming languages for describing the way a pattern is applied. In [P38, P76, and P118], the authors deal with enhancing the descriptions of design patterns. More specifically, they propose transformations for pattern application and they provide documentation on pattern usage. In [P71 and P90], constructional attributes of patterns, a comparison of pattern visualization methods, and general information on pattern comprehension are presented (Table 8).

### 5.2.2. *Design patterns detection*

During our search process, we have identified 36 articles that deal with design pattern detection. In [P29, P31, P51, and P114] the authors have created algorithms that identify behavioral and structural patterns through static and dynamic analysis. In [P4, P5, P6, and P45], the authors introduce methods for identifying structural design patterns with a multilayer approach. In [P30 and P109] introduce methods for identifying structural design patterns with a two phase approach and perform design pattern detection using a similarity scoring algorithm.

In [P32] the authors perform design pattern detection with model checking techniques. In [P27, P28, and P96] the authors identify pattern instances by visual language parsing techniques. In [P112 and P117] the authors present two pattern detection tools, for the Eiffel programming language and for UML diagrams, respectively. In [P2, P36, and P83] additional tools that discover occurrences of patterns in their standard form are presented. The [P8, P10, P68, and P100] provide techniques and tools that identify design patterns in source code.

[5] http://students.csd.auth.gr/~apamp/mapping_study.html.

**Table 8**
Keywords for pattern formalization.

| Keyword | # papers | # studies | Papers |
|---|---|---|---|
| Specification | 11 | 10 | P1, P14, P15, P37, P38, P74, P76, P85, P101, P104, P118 |
| Visualization | 6 | 5 | P35, P42, P64, P71, P88, P90, |
| Application | 4 | 4 | P19, P38, P76, P118 |
| Transformations | 4 | 4 | P19, P38, P76, P118 |
| UML | 4 | 3 | P35, P42, P64, P90 |
| Documentation | 3 | 3 | P38, P76, P118 |
| First order predicate logic | 3 | 2 | P14, P15, P104 |
| Meta-programming languages | 3 | 3 | P37, P74, P101 |
| Comprehension | 2 | 2 | P71, P88 |
| Formalization | 2 | 2 | P11, P18 |
| Formal specification | 2 | 2 | P19, P85 |
| Logical Graphs | 2 | 2 | P11, P18 |
| Architectural design artifacts | 1 | 1 | P1 |
| Class diagram | 1 | 1 | P34 |
| Collaboration diagram | 1 | 1 | P34 |
| Component based specification | 1 | 1 | P1 |
| Language | 1 | 1 | P85 |
| Repository | 1 | 1 | P19 |
| Representation | 1 | 1 | P34 |

In [P58 and P59] the authors suggest a bit-vector algorithm for pattern detection. In [P23 and P61] pattern detection algorithms have been employed for reverse engineering. In [P41 and P54] the authors aim at improving the accuracy of design pattern identification algorithms by machine learning and formal pattern specifications respectively. Paper [P88] investigates the parameters that would improve pattern detection accuracy. Paper [P43 and P113] present several benchmark problems for pattern detection. Papers [P65, P66, P89, and P102] compar pattern detection techniques and evaluate their accuracy (Table 9).

### 5.2.3. *Design patterns and software quality*

This section presents the research state of the art on the effect of GoF patterns on software quality. The primary studies are described, divided into two major categories, (1) effect on high-level quality attributes and (2) effect on low-level quality attributes.

In [P12, P13 and P16] the authors investigate testability of Abstract Factory, State, Mediator, Observer and Visitor patterns. In [P9, P33, P49, P73, P80, P91, P93 and P98] the authors deal with the maintainability, modularity, reusability, changeability and adaptability of every design pattern. More specifically, adaptability and maintainability are investigated for every pattern, whereas other quality attributes only for some of the GoF patterns.

In [P79, P99 and P110] the defect frequency arising from the use of Abstract Factory, Observer, Template Method, Adapter and Singleton are investigated. In [P39, P44, P56 and P63] the change proneness, i.e. the probability of a class to change, of Command, Composite, Decorator, Observer, Singleton, State, Factory Method, Iterator, Adapter, Bridge and Façade are investigated. In [P3, P46, P52, P55, P67, and P77] the authors focus on low-level quality attributes, such as complexity, coupling, cohesion, inheritance and size.

In [P72, P78, P79, and P81] the authors investigate the ease of adopting new requirements to instances of Factory Method, State, Visitor, Flyweight and Decorator patterns, i.e. pattern flexibility and extendibility. In [P40, P57, P62, P92, and P111] the authors investigate the maintainability, the stability and the understandability of design patterns. In [P47, P48 and P115] the understandability of Adapter, Bridge, Composite, Template Method, State, Strategy, Visitor, Bridge, Command, Observer, Proxy and Singleton patterns are investigated (Table 10).

### 5.2.4. *Design patterns application*

This section deals with the articles that investigate design pattern application. In [P50] the authors describe their experiences with design patterns. In [P17, P60 and P116] the authors suggest that design patterns are the key to provide abstraction in software and for adapting software components into existing systems. In [P20 and P75] the authors detect software anti-patterns that necessitate reengineering through design pattern application.

In [P22 and P86] focus their investigation to design patterns for Java. In [P94 and P105] propose innovative design patterns that might be a composition of other patterns. In [P69 and P70] the authors deal with generative design patterns (GDP), and try to address the major problems when using GDP. In [P24, P25, P26,

**Table 9**
Keywords for pattern detection.

| Keyword | # papers | # studies | Papers |
|---|---|---|---|
| Detection | 36 | 30 | P2, P4, P5, P6, P8, P10, P23, P27, P28, P29, P30,P31, P32, P36, P41, P43, P45, P51, P54, P58, P59, P61, P65, P66, P68, P83, P88, P89, P96, P100, P102, P109, P112, P113, P114, P117 |
| Detection Algorithm | 10 | 8 | P23, P29,P31, P41, P42, P58, P59, P61, P109, P114 |
| Detection Accuracy | 9 | 9 | P41, P42, P43, P65, P66, P88, P89, P102, P113 |
| Detection Tool | 9 | 9 | P2, P8, P10, P36, P68, P83, P100, P112, P117 |
| Detection Technique | 8 | 8 | P8, P10, P65, P66, P68, P89, P100, P102 |
| Structural | 7 | 3 | P4, P5, P6, P29, P30,P31, P114 |
| Source Code | 4 | 4 | P8, P10, P68, P100 |
| Behavioral | 3 | 2 | P29,P31, P114 |
| Dynamic Analysis | 3 | 2 | P29,P31, P114 |
| Multilayer Approach | 3 | 1 | P4, P5, P6 |
| Pattern Standard Form | 3 | 3 | P2, P36, P83 |
| Static Analysis | 3 | 2 | P29,P31, P114 |
| Benchmark | 2 | 2 | P43, P113 |
| Bit-vector | 2 | 1 | P58, P59 |
| Formal Specifications | 2 | 2 | P41, P42 |
| Language Parsing Techniques | 2 | 1 | P27, P28 |
| Machine Learning | 2 | 2 | P41, P42 |
| Reverse Engineering | 2 | 2 | P23, P61 |
| Annotation | 1 | 1 | P96 |
| Eiffel | 1 | 1 | P112 |
| Model Checking | 1 | 1 | P32 |
| Pattern Variant Form | 1 | 1 | P83 |
| Similarity Scoring | 1 | 1 | P109 |
| UML | 1 | 1 | P117 |

**Table 10**
Keywords for pattern effect on quality.

| Keyword | # papers | # studies | Papers |
|---|---|---|---|
| Quality | 10 | 10 | P16, P47, P49, P63, P73, P77, P78, P79, P91, P98 |
| Maintainability | 7 | 7 | P3, P57, P62, P67, P80, P92, P111 |
| Design Coupling | 5 | 5 | P3, P46, P52, P55, P67 |
| Stability | 5 | 5 | P39, P56, P72, P92, P110 |
| Understandability | 5 | 5 | P57, P62, P92, P111, P115 |
| Changeability | 4 | 3 | P9, P33, P44, P110 |
| Code Complexity | 3 | 3 | P3, P46, P55 |
| Defect Frequency | 3 | 3 | P79, P99, P110 |
| Flexibility | 3 | 3 | P39, P72, P78 |
| Inheritance | 3 | 3 | P46, P55, P67 |
| Modularity | 3 | 3 | P49, P62, P93 |
| Reusability | 3 | 3 | P49, P62, P93 |
| Change Proneness | 2 | 2 | P56, P63 |
| Code Cohesion | 2 | 2 | P3, P67 |
| Code Size | 2 | 2 | P3, P115 |
| Documentation | 2 | 2 | P91, P98 |
| Extendibility | 2 | 2 | P3, P81 |
| Refactoring | 2 | 2 | P77, P81 |
| Testability | 2 | 1 | P12, P13 |
| Generality | 1 | 1 | P62 |
| Open–Close principle | 1 | 1 | P80 |
| Pattern Coupling | 1 | 1 | P73 |
| Polymorphism | 1 | 1 | P52 |
| Robustness | 1 | 1 | P62 |
| Scalability | 1 | 1 | P62 |
| Usability | 1 | 1 | P40 |
| Work experience | 1 | 1 | P81 |

P53 and P82] the authors propose a methodology for automatically constructing the transformations described in design patterns. In [P106] the authors' method suggests pattern-based architecting for documenting design decisions in real-time (Table 11).

#### 5.2.5. Miscellaneous issues on design patterns

Concerning research on generic issues on GoF patterns, nine (9) studies have been identified. In [P97 and P107] the authors use Javadoc to produce HTML documentation for design patterns. In [P7, P84, P103, and P108] the use authors discuss the use of patterns in the development of a testing framework, software migration issues concerning patterns and other generic issues on GoF design patterns. In [P21, P87, and P95] refer to patterns' past, present and future, they investigate run-time behavior of several patterns and propose metrics for measuring design pattern usage intensity.

**Table 11**
Keywords for pattern application.

| Keyword | # papers | # studies | Papers |
|---|---|---|---|
| Implementation | 7 | 6 | P17, P22, P60, P69, P70, P82, P86 |
| Transformation | 5 | 3 | P24, P25, P26, P53, P82 |
| Automated Method | 4 | 2 | P24, P25, P26, P53 |
| Abstraction | 2 | 2 | P17, P60 |
| Anti-patterns | 2 | 2 | P20, P75 |
| Generative Patterns | 2 | 2 | P69, P70 |
| Java | 2 | 2 | P22, P86 |
| Pattern Composition | 2 | 2 | P94, P105 |
| Reengineering | 2 | 2 | P20, P75 |
| Component Adaptation | 1 | 1 | P116 |
| Documentation | 1 | 1 | P106 |
| Pattern-Based Architecting | 1 | 1 | P106 |
| Practical Experience | 1 | 1 | P50 |
| Programming Language | 1 | 1 | P113 |
| Real-time | 1 | 1 | P106 |

**Table 12**
Keywords for miscellaneous issues on patterns.

| Keyword | # papers | # studies | Papers |
|---|---|---|---|
| Documentation | 2 | 2 | P97, P107 |
| HTML | 2 | 2 | P97, P107 |
| Run-time Behavior | 2 | 2 | P21, P87 |
| Client Program | 1 | 1 | P84 |
| Framework | 1 | 1 | P108 |
| Pattern Application Density | 1 | 1 | P95 |
| Pattern Categorization | 1 | 1 | P103 |
| Testing Framework | 1 | 1 | P108 |

### 5.3. GoF patterns and software quality attributes

This section, presents results for further investigating the primary studies that deal with the effect of GoF design patterns on software quality attributes.

In Table 12, only the studies that deal with the effect of patterns on quality attributes are considered (in total 33 primary studies). Comparing our retrieved primary study dataset on the effect of patterns on software quality attributes with the one of Zhang and Budgen (2012) we observe that we have identified and studied 17 additional papers. These additional papers have been identified because the searching period of our review included one additional year of research, because our work did not only focus on empirical studies. However, some papers that have been reported from Zhang and Budgen (2012) are not reported in this mapping study because of the narrower searching space, in the sense that we only searched within specific journal and conference proceedings. The table summarizes how many and which studies employ which research method (as described in Glass et al. (2002)) to evaluate the effect of pattern application on software quality attributes. It is observed that the dominant empirical methodology (Wohlin et al., 2000) is *experiment*, followed by *case studies*. These results are in accordance with similar findings reported in (Glass et al., 2002; Höfer and Tichy, 2007; Zhang and Budgen, 2012) (Table 13).

Finally, Table 14 presents the reported effect of design patterns on software quality attributes and software metrics. Table 14 is presented similarly to a force resolution map as described in (Galster and Avgeriou, 2012; Souza et al., 2002). The metrics that are presented in the paper are discussed in detail in Appendix B.

In Table 14, the (+) symbol suggests that the pattern has a positive effect on the corresponding quality attribute whereas the (−) symbol suggests that the pattern has a negative effect. Finally, the study on which the selection of each symbol is based on is referenced in the brackets next to the symbol. (+) does not necessarily imply higher metric scores, but *better* metric scores. For example, a (+) in Complexity means lower complexity levels whereas a (+) in a polymorphism metric, means a system with more polymorphic behavior. (+) and (−) symbols provide guidance to researchers and practitioners, but they are by no means a strict evaluation.

The evaluation of each pattern with regard to quality attributes has been conducted according to the evaluation provided in the primary studies. Additionally, blank cells in Table 14 suggest that the corresponding pair of (*quality attribute, pattern*) has not been investigated by any study. In Table 14 we only present studies that clearly present results and discussion on the effect of a specific pattern, on a specific quality attribute. Primary studies that present more general conclusions have been presented in Section 5.2.3, but are not reported in Table 14.

## 6. Discussion

In this section we discuss the findings of our review with respect to the research questions specified in Section 4.1. In Sections 6.1 and 6.2, we summarize the current state of the art concerning design

**Table 13**
Research methods for studies on patterns and software quality.

| Name | Brief description | # papers | # studies | Papers |
|---|---|---|---|---|
| Experiment | A set of subjects is asked to perform a task in a highly controlled environment. The results are derived from observing the subjects during the experiment, from inspecting the task outcome or from questioning the subjects at the end of the procedure. | 12 (34.29%) | 12 (36.36%) | P40, P49, P57, P72, P78, P79, P80, P81, P91, P92, P98, P111 |
| Case Study | A project, an activity or an assignment is monitored with respect to the methodology under study. Results are directly derived from project measurements. | 11 (31.43%) | 10 (30.30%) | P3, P9, P16, P33, P44, P46, P56, P63, P67, P99, P110 |
| Conceptual Analysis/ Mathematical | Conceptual analysis is a technique that treats concepts as classes of objects, events, properties, or relationships. The technique involves precisely defining the meaning of a given concept by identifying and specifying the conditions under which any entity or phenomenon is classified under the concept in question. | 7 (20.00%) | 6 (18.18%) | P12, P13, P39, P52, P55, P93, P115 |
| Descriptive | A descriptive research paper typically describes a system, tool or method. | 3 (8.57%) | 3 (9.09%) | P47, P48, P77 |
| Literature Review | A literature review gathers data from already published studies synthesize them and draws colnclusions from them. | 1 (2.86%) | 1 (3.03%) | P73 |
| Survey | A set of subjects is asked to fill-in questionnaires either directly, or via internet. The results are derived from the valid answers to the questionnaire. | 1 (2.86%) | 1 (3.03%) | P62 |

patterns research. In Section 6.3, we discuss the findings of the review on the effect of patterns on high and low-level software quality attributes.

### 6.1. Research subtopics identification

The first part of analyzing existing literature with respect to research subtopics on design patterns was the introduction of a new research classification schema, which would be more fitting to design pattern research than more generic ones. Our results, pointed out five main subtopics on design pattern research, namely (a) *design pattern formalization*, (b) *design pattern detection*, (c) *effect of design patterns on software quality*, (d) *design pattern application*, and (e) *miscellaneous.*

### 6.2. Research subtopics activity

The results of Table 7 suggest that the most popular subtopics of design pattern research is design pattern detection and the investigation of the effect of patterns on software quality, followed by techniques for formulating design patterns.

Furthermore, concerning the studies that assess the effect of design pattern application on software quality, we have found out that 66.4% employ an empirical research method (Table 13). In (Glass et al., 2002; Höfer and Tichy, 2007), it is reported that in general software engineering research, i.e., not only design pattern research, the fraction of studies that employ empirical validation methods is between 20% and 30%. Although the amount of empirical research in design patterns is higher than the average empirical research in software engineering, we believe that design pattern research is in need of more empirical studies that use large-scale and realistic problems, use practitioners instead of students, use more subjects, more projects, etc. Also, maybe more diverse research methods should be applied. For example, only one survey has been conducted so far so there might be a need for more surveys. Additionally, the plethora of open-source projects, which are mainly the subjects of case studies, leaves much space for improvement both in terms of size and in the control of the case studies' environment.

### 6.3. Effect of design patterns on quality attributes

According to the results of our study (Table 14), the most commonly investigated quality attributes appear to be *maintainability, understandability*, and *reusability*. More specifically, most patterns, i.e., 18 out of 23, have been reported to have a positive effect on software maintainability. Concerning reusability, the results are controversial, because some patterns appear to provide more reusable design than others. Additionally, the understandability of design patterns seems to be the most elusive quality aspect because six patterns, i.e., *Visitor*, *Composite*, *Decorator, Proxy*, *Observer*, and *Abstract Factory*, are referenced as easily understood in some studies and as hard to understand in others. This fact can be explained by the findings of Ampatzoglou et al. (2012) where the authors identified the number of design pattern participating classes as a decider of the quality of the final design. In Ampatzoglou et al. (2012), the authors identified eleven (11) specific design pattern sizes that can be used as cut-off[6] points while comparing the understandability of the visitor pattern with alternative designs. Thus, if one case study selected a visitor instance with pattern size below one of the aforementioned cut-off points and another one a visitor instance higher than the same cut-off point, the evaluation of the two instances would be different. However, it is out of scope of this manuscript to discuss details on the structural characteristics of each pattern and attempt to explain its effect on every quality attribute. The interested reader can access such information on the primary studies that are mapped to the pair (pattern–quality attribute) he/she is interested in.

Additionally, in most cases, design patterns are about trade-offs and it is impossible to assess the effect of patterns on software quality in a generic way without knowing the context

[6] As cut-off point in (Ampatzoglou et al., 2012) the authors characterize a specific number of classes, where the metric score for a specific quality attribute, in a pattern-based version of a system, equals the metric score for a specific quality attribute, in a non-pattern-based version of the same system. If the number of classes in one system is higher than the cut-off point the one solution is better than other, if the number of classes in one system is lower than the cut-off point the other solution is better.

**Table 14**
Effect of design patterns on quality attributes.

| Attribute | Abstract Factory | Builder | Factory Method | Prototype | Singleton | Adapter | Composite | Decorator | Facade | Flyweight | Bridge | Proxy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Usability | | | | | | | | | | | | |
| Understandability | – [P040]<br>– [P062]<br>– [P111] | + [P062] | – [P062] | + [P062] | + [P062] | – [P062] | + [P062]<br>– [P111] | + [P111]<br>– [P062]<br>+ [P092] | + [P062] | – [P062] | + [P062] | + [P115]<br>– [P062] |
| Maintainability | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P072]<br>+ [P092]<br>+ [P093]<br>+ [P111] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | + [P003]<br>+ [P062]<br>– [P093] | + [P067]<br>+ [P093]<br>– [P062] |
| Stability | + [P110]<br>– [P062] | | | | – [P044] | + [P039]<br>– [P044] | – [P062] | – [P092] | + [P039] | – [P062] | + [P039] | – [P044] |
| Reliability | – [P062] | | | | | | – [P062] | | | – [P062] | | |
| Portability | | | | | | | | | | | | |
| Adaptability | + [P062]<br>+ [P093] | + [P093]<br>– [P062] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | + [P093]<br>– [P062] | – [P062]<br>– [P093] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] |
| Metrics | | | | | | | | | | | | |
| Complexity (WMC, AC) | | | | | | + [P046] | | | | | + [P003] | |
| Cohesion (H, LCOM) | | | | | | | | | | | + [P003] | – [P067] |
| Coupling (CF, Ce, CBO) | | | | | + [P046] | | | | | | + [P003] | – [P067] |
| Size (LOC, NOC) | | | | | | | | | | | – [P003]<br>– [P115] | – [P115] |
| Polymorphism (NOP) | + [P052] | + [P052] | | | | | | | | | | |
| Inheritance (DIT, NOCC, A, NMI) | | | | | | | | | | | + [P055] | – [P067] |

| Attribute | Command | Interpreter | Iterator | Mediator | Memento | Observer | State | Template method | Strategy | Visitor | Chain of responsibility |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Usability | | | | | | | | | | | |
| Understandability | – [P062] | + [P062] | + [P062] | + [P062] | – [P062] | + [P111]<br>– [P062]<br>– [P092] | + [P062] | – [P062] | + [P062] | + [P062]<br>– [P051]<br>– [P058]<br>– [P111] | + [P062] |
| Maintainability | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] | + [P003]<br>+ [P062]<br>– [P093] | + [P062]<br>+ [P093] | + [P062]<br>+ [P093] | + [P058]<br>+ [P062]<br>+ [P072]<br>+ [P092]<br>+ [P093]<br>– [P051]<br>– [P111] | + [P062]<br>+ [P093] |
| Stability | + [P044] | | | | | – [P110] | – [P044] | + [P110] | – [P044] | | |
| Testability | | | | – [P013] | | – [P013] | | | | – [P013] | |
| Portability | | | | | | | | | | | |
| Adaptability | + [P093]<br>– [P062] | – [P093] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] | + [P093]<br>– [P062] | – [P062]<br>– [P093] | + [P062]<br>+ [P093] |
| Metrics | | | | | | | | | | | |
| Complexity (WMC, AC) | | | | | | | + [P003]<br>– [P003] | | | + [P055]<br>– [P046] | |
| Cohesion (H, LCOM) | | | | | | | + [P003] | | | | |
| Coupling (CF, Ce, CBO) | | | | + [P052]<br>+ [P055] | | + [P052]<br>+ [P046] | + [P003] | | | + [P046] | |
| Size (LOC, NOC) | – [P115] | | | | | – [P115] | – [P003] | | | | |
| Polymorphism (NOP) | | | | | | + [P052] | + [P052] | | + [P052] | | |
| Inheritance (DIT, NOCC, A, NMI) | | | | | | | | | | + [P046] | |

(flexibility, requirements, etc.). Most pattern catalogs describe how to introduce flexibility, but as a side-effect, they introduce more complexity. This is preferable in cases when extra flexibility is required. If a design pattern leads to more complexity to enhance system flexibility, without really needing it, software quality would deteriorate. Usually, the application of one pattern enhances some quality attributes and simultaneously other quality attributes diminish. For example, the Abstract Factory pattern is reported to be beneficial to maintainability, because new product types can be added without altering the existing code.However, the application of the Abstract Factory also decreases the understandability of the design. The designer should consider which quality attribute is more important to him/her and decide if he/she will apply the pattern.

Additionally, although design patterns are about design, most existing studies consider the implementation of the solutions of design patterns in source code for practical reasons. Furthermore, implementation of a design pattern can vary across primary studies. Thus, we believe that these variants themselves could be responsible for any difference observed in the effect of pattern usage on low-level and high-level quality attributes.

As the current state of the art stands, evaluation of design patterns has been performed through code metrics in only six studies. In the rest of the cases, the evaluation has been performed by expert opinion. Although evaluating patterns through metrics is not necessarily superior than evaluating according to expert judgment, assessing the effect of patterns through code and design metrics is a field that needs further investigation and is expected to grow in the next years. This fact is explained by the findings of Zhang et al., where it is suggested that the majority of empirical studies that investigate design patterns are conducted by controlled experiments (Zhang and Budgen, 2012). Additionally, pattern coupling is clearly mentioned in only two studies, although in real systems pattern occurrences interact. Thus, evaluating isolated pattern instances may be risky or just an approach of the real effect of the pattern on real systems.

According to the results of the study, we suggest that future research might focus on areas such as pattern understandability, pattern reusability, and reusability of patterns that are used at the component or subsystem levels. Additionally, we believe that interesting future work might deal with identifying variables that would formulate the abovementioned trade-offs and be used in decision making tools that would evaluate pattern application benefits and drawbacks. Such tools would help practitioners during system design in deciding whether to use a pattern or not. Furthermore, we propose future research efforts to deal with the effect of patterns on quality attributes such as usability, modularity, generality, scalability, and robustness, which have not been thoroughly investigated yet. Moreover, we claim that interesting future work would be the evaluation of design patterns at the design level rather than on an implementation level, that is only considered at this point. Finally, studies that will investigate real systems with pattern instances that interact appear to be of great interest.

## 7. Threats to validity

In this section, we discuss possible threats to the validity of our study. Threats to validity are divided into four major categories, namely (a) construct validity threats, (b) internal validity threats, (c) external validity threats and (d) threats to conclusions validity.

### 7.1. Construct threats to validity

Construct threats to validity deal with problems that might arise during research design. In a literature review, such threats are related to the *identification of primary studies*. Concerning our search process, any study that does not mention the word “pattern” in the title of the article has been excluded from the primary studies set. So, a number of articles that deal with design patterns might have been omitted. However, we believe that papers that deal with design patterns would most probably explicitly state it in their titles. Additionally, not performing a global search on an indexing system such as SCOPUS, EI COMPENDIX, or Web of Science, means that papers in less popular journals and conferences may have been omitted from the study. However, we believe that including in the review only top journals, conference, and workshops, raise the quality standards of the primary studies and therefore the quality of the results of our systematic review. This is in line with past published surveys similar to our work. We have chosen not to include PLoP conferences in the searched venues, because the studies published in PLoP are usually about introducing new patterns and not discussing the GoF patterns. The acceptance rate of PLoP conferences could not be retrieved to validate if it fits the selection criteria for conferences.

### 7.2. Internal threats to validity

Internal threats to validity deal with problems that arise during *data extraction*. Concerning the results on the impact of design patterns on software quality, we identify three limitations. Firstly, although we tried to map every quality attribute that was identified in a primary study to the ISO 9126 quality model, there might be the case of a misplaced study, because sometimes researchers have different understandings of quality attributes and use the same term for different attributes or different terms for the same quality attribute. However, it was not possible to locate a single quality model that references all quality attributes that are used in the primary studies. Additionally, using a quality model and merging quality attributes in larger categories has been preferred over presenting the quality attributes as described in the primary studies, because primary studies have used different names for similar or same quality attributes and this fact leads to loss of data synthesis and categorization opportunities. Secondly, design patterns are blue prints that leave variants implementation options to the developers. Considering that implementation plays important role with respect to quality attributes, there might be possible deviations from the metrics calculated in the primary studies that we reference. However, the majority of primary studies investigated the standard pattern implementations (Gamma et al., 1995), consequently the results of the study deal with the standard pattern forms and cannot be generalized to pattern variations or implementation different from the original implementation (Gamma et al., 1995). Finally, the extracted data concerning the keywords, quality attributes and research methods of primary studies have been identified through expert judgment and therefore they are no strict characterizations of the papers. However, the way that conflicts have been resolved limits the possibilities of errors.

### 7.3. External threats to validity

External threats to validity deal with problems, in generalizing the obtained results from the sample to the population. The conclusions that have been drawn are design pattern type specific. Thus, results on one pattern for which a lot of data are available cannot be generazlized to other design patterns for which we do not have enough data. In addition to this, results from primary studies which are mainly concerned with maintenance tasks results cannot be generalized to development tasks.

### 7.4. Threats to conclusions validity

Threats to conclusions validity are factors that can lead to incorrect conclusions, either by identifying incorrect relationship, or by missing existing relationships. In the case of our study, such factors are related to identification of primary studies, i.e., missing studies that should have been included in the review, and incorrect data extraction. Both these threats are discussed in detail in the previous paragraphs. An additional threat to validity is that we measure research intensity with the number of studies and not with the volume of research or information they provide. However, such an attempt would introduce subjective criteria into the analysis of the results. For this reason, we preferred to use an objective (directly measurable) criterion for characterizing the intensity of research. Finally, having not merged multiple papers under one study, might have slightly altered our results. However, we believe that in the common case, major journals and conferences, such as the ones we have used as a searching space rarely publish papers that are not based on innovative ideas and studies.

## 8. Conclusions

This paper aims at summarizing the research state of the art on GoF design patterns, emphasizing on studies that deal with the effect of pattern application on quality attributes. The main research questions that the mapping study answers are: (a) if design pattern research can be further categorized in research subtopics, (b) which of the above subtopics are the most active and (c) what is the reported effect of GoF patterns on software quality attribute.

The research efforts on GoF design patterns have been classified according a classification schema, which has been extracted according to our search result, into five subtopics: (a) papers on pattern formalizations, i.e. papers that attempt to provide formal descriptions of design patterns, (b) papers on pattern detection, i.e. papers that propose algorithms/tools/techniques for identifying design pattern instances, (c) papers on pattern application, i.e. papers that discuss how design patterns can be applied and implemented, (d) papers on the effect of patterns on software quality, i.e. papers that evaluate the effect of GoF patterns on software quality attributes, and (e) miscellaneous papers, i.e. papers that cannot be placed in any other subtopic.

The most active research subtopics are design pattern detection, and the impact of GoF patterns on software quality attributes. The main bulk of publications that evaluate the effect of patterns on software quality consist of empirical studies that appear to investigate both low-level and high-level software quality attributes. Until now, the research efforts produce controversial results and practitioners must employ judgment to select the most fitting design. Additionally, the results confirm that patterns are about trade-offs. Design patterns enhance one quality attribute in the expense of another. Consequently, design patterns cannot be characterized as universally "good" or "bad", but the quality attributes that are important for the specific application must be examined as well.

The abovementioned indications suggest that in the near future, researchers might attempt to narrow these gaps by employing research approaches other than empirical methods, to investigate the effect of pattern application on software quality attributes, e.g., by analytical methods as in [2,P52, and P55]. However, we believe that research on both design patterns and software engineering in general are still in need of empirical studies. Furthermore, this mapping study pointed out several gaps in the research state of the art concerning the effect of patterns on software quality, as summarized below:

- Design pattern understandability
- Design pattern reusability
- Design pattern modularity
- Design pattern robustness
- Design pattern coupling

**Acknowledgements**

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thalis–Athens University of Economics and Business – SOFTWARE ENGINEERING RESEARCH PLATFORM. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. Especially their comments on the correct application of the methodology and the distinction of papers and studies have been extremely valuable for us.

## Appendix A. Papers included in the review

[P1] P.S.C. Alencar, D.D. Cowan, J. Dong and C.J.P. de Lucena, "A Pattern-Based Approach to Structural Design Composition", *23rd International Computer Software and Applications Conference(COMPSAC'99)*, IEEE, pp. 160–165, Phoenix, Arizona, 25–26 October 1999.

[P2] H. A. Amiot, P. Cointe, Y. G. Guéhéneuc and N. Jussien, "Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together", *Proceedings of the 16th IEEE international conference on Automated software engineering*, ACM, pp. 166, San Diego, California, 26–29 November 2001.

[P3] 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.

[P4] G. Antoniol, G. Casazza, M. Di Penta and R. Fiutem, "Object-Oriented design patterns recovery", *Journal of Systems and Software*, Elsevier, 59 (2), pp. 181–196, November 2001.

[P5] G. Antoniol, R. Fiutem and L. Christoforetti, "Using Metrics to Identify Design Patterns in Object-Oriented Software", *Proceedings of the 5th International Symposium on Software Metrics*, IEEE, pp. 23, Bethesda, Maryland, 20–21 March 1998.

[P6] G. Antoniol, R. Fiutem and L. Christoforetti, "Design Pattern Recovery in Object-Oriented Systems", *Proceedings of the 6th International Conference on Program Comprehension* (ICPC' 98), IEEE, Ischia, Italy, 24–26 July 1998.

[P7] F. Arcelli, C. Tosi and M. Zanoni, "Can design pattern detection be useful for legacy systemmigration toward SOA?", *Proceedings of the 2nd international workshop on Systems development in SOA environments(ICSE' 08)*, IEEE, pp. 63–68, Leipzig, Germany, 10–18 May 2008.

[P8] A. Asencio, S. Cardman, D. Harris and E. Laderman, "Relating Expectations to Automatically Recovered Design Patterns", *Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02)*, pp. 87, Richmond, Virginia, 29 October–01 November 2002.

[P9] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso and M. Di Penta, "An empirical study on the evolution of design patterns", *Foundations of Software Engineering (FSE' 07)*, ACM, pp. 385–394, Dubrovnik, Croatia, 3–7 September 2007.

[P10] Z. Balanyi and R. Ferenc, "Mining Design Patterns from C++ Source Code", *Proceedings of the International Conference on Software Maintenance*, IEEE, Amsterdam, The Netherlands, 22–26 September 2003.

[P11] E. L. A. Baniassad, G. C. Murphy and C. Schwanninger, "Design Pattern Rationale Graphs: linking design to source", *Proceedings of the 25th International Conference on Software Engineering*, IEEE, pp. 352–362, Portland, Oregon, 03–10 May 2003.

[P12] B. Baudry, Y. Le Sunye and J. M. Jezequel, "Toward 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.

[P13] B. Baudry, Y. Le Traon, G. Sunyè and J. M. Jezequel, "Measuring and Improving Design Patterns Testability", *Proceedings of the 9th International Symposium on Software Metrics*, IEEE, pp. 50, Sydney, Australia, 03–05 September 2003.

[P14] I. Bayley and H. Zhu, "Formal specification of the variants and behavioral features of design patterns", *Journal of Systems and Software*, Elsevier, 83 (2), pp. 209–221, February 2010.

[P15] I. Bayley and H. Zhu, "Specifying Behavioral Features of Design Patterns in First Order Logic", *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC '08)*, IEEE, pp. 203–210, Turku, Finland, 28 July-01 August 2008.

[P16] J. M. Bieman, G. Straw, H. Wang, P. W. Munger and R. T. Alexander, "Design Patterns and Change Proneness: An Examination of Five Evolving Systems", *Proceedings of the 9th International Symposium on Software Metrics*, IEEE, pp. 40, Sydney, Australia, 03–05 September 2003.

[P17] J. Bishop, "Language features meet design patterns: raising the abstraction bar", *Proceedings of the 2nd international workshop on The role of abstraction in software engineering (ICSE'08)*, IEEE, pp. 1–7, Leipzig, Germany, 10–18 May 2008.

[P18] A. Blewitt, A. Bundy and I. Stark, "Automatic verification of design patterns in Java", *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ACM, pp. 224–232, Long Beach, CA, 07–11 November 2005.

[P19] G. E. Boussaidi and H. Mili, "A model driven framework for representing and applying design patterns", *31st Annual International Computer Software and Applications Conference (COMPSAC'07)*, IEEE, pp 97–100, Beijing, China, 24–27 July 2007.

[P20] L. C. Briannd, Y. Labiche and A. Sauve, "Guiding the Application of Design Patterns Based on UML Models", *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, IEEE, pp. 234–243, Philadelphia, Pennsylvania, 24–27 September 2006.

[P21] F. Buschmann, K. Henney and D. C. Schmidt, "Past, Present and Future in Software Patterns", *IEEE Software*, IEEE, 24 (4), pp. 31–37, July 2007.

[P22] M. I. Cagnin, R. T. V. Braga, P. C. Masiero, I. Usp, R. Penteado, and Dc UFSCar, "Reengineering using Design Patterns", *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, pp.118, Brisbane, Australia, 23–25 November 2000.

[P23] R. Chaabane, "Poor performing patterns of code: Analysis and Detection", *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'07)*, IEEE, pp. 501–502, Paris, France, 02–05 October 2007.

[P24] M. O' Cinneide, "Automated refactoring to introduce design patterns", *Proceedings of the 22nd international conference on Software engineering (ICSE'00)*, IEEE, pp. 722–724, Limeric, Ireland, 4–11 June 2000.

[P25] M. O' Cinneide and P. Nixon, "A Methodology for the Automated Introduction of Design Patterns", *Proceedings of the 15th IEEE International Conference on Software Maintenance*, IEEE, pp. 463, Oxford, England, 30 August–03 September 1999.

[P26] M. O' Cinneide and P. Nixon, "Automated software evolution toward design patterns", *Proceedings of the 4th International Workshop on Principles of Software Evolution (ICSE'01)*, IEEE, pp. 162–165, Vienna, Austria, 12–19 May 2001.

[P27] G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "Case Studies of Visual Language Based Design Patterns Recovery", *Proceedings of the Conference on Software Maintenance and Reengineering*, IEEE, pp. 165–174, Bari, Italy, 22–24 March 2006.

[P28] G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "Design Pattern Recovery by Visual Language Parsing", *Proceedings of the 29th international conference on Software Engineering*, IEEE, pp 102–111, Manchester, UK, 21–23 March 2005.

[P29] A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "Design pattern recovery through visual language parsing and source code analysis", *Journal of Systems and Software*, 82(7), pp. 1177–1193, July 2009.

[P30] A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "A Two Phase Approach to Design Pattern Recovery", *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, IEEE, pp. 297–306, Amsterdam, the Netherlands, 21–23 March 2007.

[P31] A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation", *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, IEEE, pp. 99–108, Kaiserslautern, Germany, 24–27 March 2009.

[P32] A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "Improving Behavioral Design Pattern Detection through Model Checking", *Proceedings of IEEE European Conference on Software Maintenance and Reengineering (CSMR'10)*, IEEE press, pp. 176–185, Madrid, Spagna, 15–18 March, 2010.

[P33] M. Di Penta, L. Cerulo, Y. G. Guéhéneuc and G. Antoniol, "An empirical study of the relationships between design pattern roles and class change proneness", *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'08)*, IEEE, pp. 217–226, Beijing, China, 28 September–04 October 2008.

[P34] J. Dong, "Adding pattern related information in structural and behavioral diagrams", *Information and Software Technology*, Elsevier, 46 (5), pp. 293–300, 15 April 2004.

[P35] J. Dong, S. Yang and K. Zhang, "Visualizing Design Patterns in Their Applications and Compositions", *IEEE Transactions on Software Engineering*, IEEE, 33 (7), pp. 433–453, July 2007.

[P36] J. Dong and Y. Zhao, "Experiments on Design Pattern Discovery", *Proceedings of the Third International Workshop on Predictor Models in Software Engineering (ICSE'07)*, IEEE, pp. 12, Minneapolis, Minnesota, 23–25 May 2007.

[P37] A. H. Eden, A. Yehudai, and J. Gil, "Precise specification on automatic application of design patterns", *Proceedings of the 12th international conference on Automated software engineering (formerly: KBSE)*, ACM, pp. 143, Lake Tahoe, CA, 02–05 November 1997.

[P38] E. Eide, A. Reid, J. Regehr and J. Lepreau, "Static and Dynamic structure in design patterns", *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)*, IEEE, pp. 208–218, Orlando, Florida, 19–25 May 2002.

[P39] M. Elish, "Do Structural Design Patterns Promote Design Stability?", *Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 01 (COMPSAC'06)*, IEEE, pp 215–220, Chicago, Illinois, 17–21 September 2006.

[P40] 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.

[P41] R. Ferenc, A. Beszedes, L. Fulop and J. Lele, "Design Pattern Mining Enhanced by Machine Learning", *Proceedings of the 21st IEEE International Conference on Software Maintenance*, IEEE, pp. 295–304, Budapest, Hungary, 25–30 September 2005.

[P42] R. B. France, D. K. Kim, S. Ghosh and E. Song, “A UML-Based Pattern Specification Technique”, *IEEE Transactions on Software Engineering*, IEEE, 30 (3), pp. 193–206, March 2004.
[P43] L. J. Fulop, R. Ferenc and T. Gyimothy, “Toward a Benchmark for Evaluating Design Pattern Miner Tools”, *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, IEEE, pp. 143–152, Athens, Greece, 01–04 April 2008.
[P44] M. Gatrell, S. Counsell and T. Hall, “Design Patterns and Change Proneness: A Replication Using Proprietary C# Software”, *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, pp. 160–164, Lille, France, 13–16 October 2009.
[P45] Y.-G. Guéhéneuc and G. Antoniol, “DeMIMA: A Multilayered Approach foe Design Pattern Identification”, *IEEE Transaction of Software Engineering*, IEEE, 34 (5), pp. 667–684, September 2008.
[P46] Y. -G. Guéhéneuc, H. Sahraoui and F. Zaidi, “Fingerprinting Design Patterns”, *Proceedings of the 11th Working Conference on Reverse Engineering*, pp. 172–181, Delft, The Netherlands, 08–12 November 2004.
[P47] J. Gustafsson, J. Paakki, L. Nenonen and A. I. Verkamo, “Architecture-Centric Software Evolution by Software Metrics and Design Patterns”, *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, IEEE, pp. 108, Budapest, Hungary, 11–13 March 2002.
[P48] Z. Han, L. Wang, L. Yu, X. Chen, J. Zhao, X. Li, “Design pattern directed clustering for understanding open source code”, *International Conference on Program Comprehension* (ICPC 09), IEEE, pp. 295–296, Vancouver, British Columbia, Canada, 17–19 May 2009.
[P49] J. Hannemann and G. Kiczales. “Design Pattern Implementation in Java and AspectJ”, Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications(OOPSLA ‘02), ACM, pp. 161–173, Seattle, Washington, 4–8 November 2002.
[P50] R. Helm, “Patterns in Practice”, Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications (OOPSLA ‘95), ACM, pp. 337–341, Austin, Texas, 15–19 October 1995.
[P51] D. Heuzeroth, T. Holl, G. Högström, W. Löwe, “Automatic Design Pattern Detection”, Proceedings of the 11th International Working Conference on Program Comprehension (ICPC ‘03), IEEE, Portland, USA, 10–11 May 2003.
[P52] 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.
[P53] N.L. Hsueh, P.H. Chu, P.A. Hsiung, M.J. Chuang, W. Chu, C.H. Chang, C.S. Koong and C.H. Shih, “Supporting Design Enhancement by Pattern-Based Transformation”, *34th Annual Computer Software and Applications Conference (COMPSAC ‘10), IEEE*, pp. 462–467, Seoul, Korea, 19–23 July 2010.
[P54] H. Huang, S. Zhang, J. Cao, Y. Duan, “A practical pattern recovery approach based on both structural and behavioral analysis”, *Journal of Systems and Software*, Elsevier, 75 (1–2), pp. 69–87, February 2005.
[P55] B. Huston, “The effects of design pattern application on metric scores”, *Journal of Systems and Software*, Elsevier, 58 (3), pp. 261–269, September 2001.
[P56] D. Jain and H. J. Yang, “OO Design Patterns, Design Structure, and Program Changes: An Industrial Case Study” *Proceedings of the IEEE International Conference on Software Maintenance (ICSM’01)*, IEEE, pp. 580, Florence, Italy, 07–09 November 2003.
[P57] S. Jeanmart, Y. -G. Guéhéneuc, H. Sahraoui and N. Habra, “A Study of the Impact of the Visitor Design Pattern on Program Comprehension and Maintenance Tasks”, *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* (*ESEM*’ *09)*, IEEE, pp. 69–78, Lake Buena Vista, Florida, 15–16 October 2009.
[P58] O. Kaczor, Y.-G. Guéhéneuc, and S. Hamel, “Efficient Identification of Design Patterns with Bit-vector Algorithm”, *IEEE Proceedings of the 10th Conference on Software Maintenance and Reengineering, (CSMR’06)*, IEEE, pp. 175–184, 22–24 March 2006.
[P59] O. Kaczor, Y. -G. Guéhéneuc and S. Hamel, “Identification of design motifs with pattern matching algorithms”, *Information and Software Technology*, Elsevier, 52 (2), pp. 152–168, February 2010.
[P60] B. Keepence and M. Mannion, “Using Patterns to Model Variability in Produst Families”, *IEEE Software*, IEEE, 16 (4), pp. 102–108, July 1999.
[P61] R. K. Keller, R. Schauer, S. Robitaille and P. Page, “Pattern-based reverse-engineering of design components”, *Proceedings of the 21st international conference on Software engineering (ICSE’99)*, IEEE, pp. 226–235, Los Angeles, California, 16–22 May 1999.
[P62] F. Khomh and Y.-G. Guéhéneuc, “Do Design Patterns Impact Software Quality Positively?”**,** *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, IEEE, pp. 274–278, Athens, Greece, 01–04 April 2008.
[P63] F. Khomh and Y.-G. Guéhéneuc, “Playing roles in design patterns: An empirical descriptive and analytic study”, *Proceedings of the 25th IEEE International Conference on Software Maintenance*, IEEE, pp 83–92, Edmonton, Canada, 20–26 September 2009.
[P64] D. K. Kim, R. France, S. Ghosh and E. Song, “A Role-Based Metamodeling Approach to Specifying Design Patterns”, *Proceedings of the 27th Annual International Conference on Computer Software and Applications*, IEEE, pp. 452, Dallas, Texas, 03–06 November 2003
[P65] G. Kniesel, A. Binun, “Standing on the shoulders of giants – A data fusion approach to design pattern detection”, *International Conference on Program Comprehension* (ICPC’ 09), IEEE, pp. 208–217, Vancouver, British Columbia, Canada, 17–19 May 2009.
[P66] G. Kniesel, A. Binun, P. Hegedus, L. J. Fülöp, A. Chatzigeorgiou, Y.G. Guéhéneuc, and N. Tsantalis, “DPDX – A Common Exchange Format for Design Pattern Detection Tools”, *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR’10)*, IEEE, p.p 232–235, Madrid, Spain, 15–18 March 2010.
[P67] K. Kouskouras, A. Chatzigeorgiou and G. Stephanides, “Facilitating software extension with design patterns and Aspect-Oriented Programming”, *Journal of Systems and Software*, Elsevier, 81 (10), pp 1725–1737, October 2008.
[P68] C. Kramer and L. Prechelt, “Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software”, *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE ‘96)*, IEEE, pp. 208, Monterey, CA, 8–10 November 1996.
[P69] S. MacDonald, D. Szafron, J. Schaeffer, J. Anvik, S. Bromling and K. Tan, “Generative Design Patterns”, *Proceedings of the 17th IEEE international conference on Automated software engineering*, ACM, pp. 23, Edinburgh, UK, 23–27 September 2002.
[P70] S. MacDonald, K. Tan, J. Schaeffer and D. Szafron, “Deferring Design Pattern Decisions and Automating Structural Pattern Changes Using a Design-Pattern-Based Programming System”, *ACM Transactions on Programming Languages and Systems*, ACM, 31(3), article 9, April 2009.
[P71] J. K. H. Mak, C. S. T. Choy and D. P. K. Lun, “Precise Modeling of Design Patterns in UML”, *Proceedings of the 26th International Conference on Software Engineering (ICSE’04)*, IEEE, pp. 252–261, Edimburg, Scotland, 23–28 May 2004.
[P72] B.A. Malloy and J.F. Power, “Exploiting design patterns to automate validation of class invariants: Research articles”, *Software Testing Verification & Reliability*, Wiley Interscience, 16 (2), pp. 71–95, June 2006.
[P73] W. B. McNatt and J. M. Bieman, “Coupling of Design patterns: Common Practices and Their Benefits”, *25th Annual International*

*Computer Software and Applications Conference (COMPSAC'01)*, IEEE, pp.574, Chicago, Illinois, 08–12 October 2001.
[P74] T. Mens and T. Tourwe, "A Declarative Evolution Framework for Object-Oriented Design Patterns", *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, IEEE, pp. 570, Florence, Italy, 07–09 November 2003.
[P75] M. Meyer, "Pattern-based Reengineering of Software Systems", *Proceedings of the 13th Working Conference on Reverse Engineering*, pp. 305–306, Benevento, Italy, 23–27 October 2006.
[P76] T. Mikkonen, "Formalizing design patterns", *Proceedings of the 20th international conference on Software engineering (ICSE'98)*, IEEE, pp. 115–124, Kyoto, Japan, 19–25 April 1998.
[P77] 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.
[P78] T. H. Ng and S. C. Cheung, "Enhancing class commutability in the deployment of design patterns", *Information and Software Technology*, Elsevier, 47 (12), pp. 797–804, September 2005.
[P79] T. H. Ng, S. C. Cheung, W. K. Chan and Y. T. Yu, "Do Maintainers Utilize Deployed Design Patterns Effectively?", *International Conference on Software Engineering*, IEEE, pp. 168–177, Minneapolis, Minnesota, 20–26 May 2007.
[P80] T. H. Ng, S. C. Cheung, W. K. Chan and Y. T. Yu, "Toward effective deployment of design patterns for software extension: a case study", *Proceedings of the 2006 international workshop on Software quality (ICSE'06*), IEEE, pp. 51–56, Shanghai, China, 21 May 2006.
[P81] T. H. Ng, S. C. Cheung, W. K. Chan and Y. T. Yu, "Work experience versus refactoring to design patterns: a controlled experiment" *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ACM, pp. 12–22, Portland, Oregon, 5–11 November 2006.
[P82] S.J. Nielson and C. D. Knutson, "Design dysphasia and the pattern maintenance cycle", *Information and Software Technology*, Elsevier, 48 (8), pp. 660–675, August 2006.
[P83] J. Niere, W. Schafer, J. P. Wadsack, L. Wendehals and J. Welsh, "Toward pattern-based design recovery", *Proceedings of the 24th International Conference on Software Engineering*, IEEE, pp. 338–348, Orlando, Florida, 19–25 May 2002.
[P84] N. Noda and T. Kishi, "Design pattern concerns for software evolution", *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IEEE, pp. 158–161, Vienna, Austria, 10–11 September 2001.
[P85] B. C. d. S. Oliveira, M. Wang and J. Gibbons, "The Visitor Pattern as a Reusable, Generic, Type-Safe Component", Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications (OOPSLA 2008), ACM, pp. 439–456, Nashville, Tennessee, 19–23 October 2008.
[P86] J. Palsberg and C. B. Jay, "The Essence of the Visitor Pattern", *Proceedings of the 22nd International Computer Software and Applications Conference*, IEEE, pp. 9–15, Vienna, Austria, 17–21 August 1998.
[P87] C. Park, Y. Kang, C. Wu and K. Yi, "A Static Reference Flow Analysis to Understand Design Pattern Behavior" *11th Working Conference on Reverse Engineering (WCRE 2004)*, pp. 300–301, Delft, The Netherlands, 08–12 November 2004.
[P88] N. Pettersson, "Measuring precision for static and dynamic design pattern recognition as a function of coverage", *Proceedings of the third international workshop on Dynamic analysis (ICSE'05*), IEEE, pp. 1–7, St. Louis, Missouri, 17 May 2005.
[P89] N. Petterson, W. Löwe and J. Nivre, "Evaluation of Accuracy in Design Pattern Occurrence Detection", *IEEE Transactions on Software Engineering*, IEEE, 36 (4), pp. 575–590, July/ August 2010.
[P90] G. C. Porras and Y. G. Guéhéneuc, "An Empirical Study on the Efficiency of Different Design Pattern Representations in UML Class Diagrams", *Empirical Software Engineering*, Springer, Jan 2010.
[P91] L. Prechelt, B. Unger-Lamprecht, M. Philipsen and W. F. Tichy, "Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance", *IEEE Transactions on Software Engineering*, IEEE, 28 (6), pp. 595–606, June 2002.
[P92] 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.
[P93] H. Rajan, S. M. kautz and W. Rowcliffe, "Concurrency by Modularity: Design Patterns, a Case in Point", Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '10), ACM, pp. 790–805, Reno, Nevada, 17–21 October 2010.
[P94] D. Riehle "Composite Design Patterns", Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97), ACM Press, pp. 218–228, Atlanta, Georgia, 05–09 October 1997.
[P95] D. Riehle "Design Pattern Density Defined", Proceeding of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications (OOPSLA '09), ACM, pp. 469–480, Orlando, Florida, 25–29 October 2009.
[P96] G. Rasool, I. Philippow, P. Mader, "Design pattern recovery based on annotations", *Advances in Engineering Software*, Elsevier, 41(4), pp. 519–526, April 2009.
[P97] J. Sametinger and M. Riebish, "Evolution Support by Homogeneously Documenting Patterns, Aspects and Traces", *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, IEEE, pp. 134, Budapest, Hungary, 11–13 March 2002.
[P98] G. Scanniello, C. Gravino, M. Risi and G. Tortora, "A controlled experiment for assessing the contribution of design pattern documentation on software maintenance", *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement(ESEM' 10)*, ACM, Bolzano-Bozen, Italy, 16–17 September 2010.
[P99] T. Schanz and C. Izurieta, "Object Oriented Design Pattern Decay: A Taxonomy", *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (*ESEM*' *10)*, ACM, Bolzano-Bozen, Italy, 16–17 September 2010.
[P100] N. Shi and R. A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code", *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, ACM, pp. 123–134, Tokyo, Japan, 18–22 September 2006.
[P101] N. Soundarajan and J. O. Hallstrom, "Responsibilities and Rewards: Specifying Design patterns", *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, IEEE, pp. 666–675, Edinburgh, UK, 23–28 May 2004.
[P102] D. Streitferdt, C. Heller and I. Philippow, "Searching Design Patterns in Source Code", *Proceedings of the 29th Annual International Computer Software and Applications Conference – Volume 02 (COMPSAC'05)*, IEEE, pp. 33–34, Edinburgh, UK, 26–28 July 2005.
[P103] L. Tahvildari and K. Kontogiannis, "On the Role of Design Patterns in Quality-Driven Re-engineering", *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, IEEE, pp. 134, Budapest, Hungary, 11–13 March 2002.
[P104] T. Taibi and D.C. L. Ngo, "Formal specification of design pattern combination using BPSL", *Information and Software Technology*, Elsevier, 45 (3), pp. 157–170, March 2003.
[P105] T. D. Thu and H. T. B. Tran, "A Composite Design Pattern for Object Frameworks", *31st Annual International Computer Software*

*and Applications Conference (COMPSAC'07)*, IEEE, pp. 521–526, Beijing, China, 24–27 July 2007.

[P106] P. Tonella and G. Antoniol, "Object Oriented Design Pattern Inference", *Journal of Software Maintenance*, Wiley, 13 (5), September-October 2001.

[P107] M. Torchiano, "Documenting Pattern Use in Java Programs", *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, IEEE, Montreal, Canada, 03–06 October 2002.

[P108] W. T. Tsai, Y. Tu, W. Shao and E. Ebner, "Testing Extensible Design Patterns in Object Oriented Frameworks through Scenario Templates", *23rd International Computer Software and Applications Conference (COMPSAC'99)*, IEEE, pp. 166–171, Phoenix, Arizona, 25–26 October 1999.

[P109] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides and S. T. Halkidis, "Design Pattern Detection Using Similarity Scoring", *IEEE Transaction of Software Engineering*, IEEE, 32 (11), pp. 896–909, November 2006.

[P110] M. Vokáč, "Defect Frequency and Design Patterns: An Empirical Study of Industrial Code", *IEEE Transactions on Software Engineering*, IEEE, 30(12), pp. 904–917, December 2004.

[P111] 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.

[P112] W. Wang and V. Tzerpos, "Design Pattern Detection in Eiffel Systems", *Proceedings of the 12th Working Conference on Reverse Engineering*, pp. 165–174, Pittsburgh, Pennsylvania, 07–11 November 2005.

[P113] P. Wegrzynowicz and K. Stencel, "Toward a comprehensive Test Suite for Detectors of Design Patterns", *24th IEEE/ACM International Conference on Automated Software Engineering (ASE '09)*, pp. 103–110, Auckland, New Zealand, 16–20 November 2009.

[P114] L. Wendehals and A. Orso, "Recognizing behavioral patterns at run time using finite automata", *Proceedings of the 2006 international workshop on Dynamic systems analysis (ICSE'06)*, IEEE, pp. 33–40, Shanghai, China, 23 May 2006.

[P115] P. Wendorff, "Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project", *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, IEEE, pp. 77, Lisbon, Portugal, 14–16 March 2001.

[P116] S. S. Yau and N. Dong, "Integration in Component-Based Software Developmant Using Design Patterns", *24th International Computer Software and Applications Conference (COMPSAC'00)*, IEEE, pp. 369, Taipei, Taiwan, 25–28 October 2000.

[P117] H. Zhu, I. Bayley, L. Shan and R. Amphlett, "Tool Support for Design Pattern Recognition at Model Level", *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference*, IEEE, pp. 228–233, Seattle, Washington, 20–24 July 2009.

[P118] M. Ziane, "A Transformational Viewpoint on Design Patterns", *Proceedings of the 15th IEEE international conference on Automated software engineering*, ACM, pp. 273, Grenoble, France, 11–15 September 2000.

## Appendix B. Low-level quality metrics

**A** (*Abstractness*)
Abstractness is the ratio of abstract classes and interfaces to the total number of types (classes, concrete or abstract, and interfaces) of the measured package. By definition, abstract classes and interfaces are certain to draw incoming dependencies. Then, in combination with the discussion on instability above, packages with high abstractness would be good to be stable (and vice versa).

**AC** (*Attribute Complexity*)
Is defined as the sum of each attribute's value in the class. You can set up weights for types and their arrays separately under Attribute type complexity. Use "*" to define types of a package with all its subpackages. For example, java.lang.* means that the row defines all classes of the java.lang package and its subpackages. To process all types not listed in the table, specify the last row as "*". The row order is important, because checking of attributes goes from the top of the table downward. (Repetitions of a type aren't counted, so if a specific type follows a more general type that already included it, the specific type isn't counted. For example, java.lang.* won't be counted if it comes after java.*)

**CBO** (*Coupling Between Objects*)
Represents the number of other classes to which a class is coupled. Counts the number of reference types that are used in attribute declarations, formal parameters, return types, throws declarations and local variables, and types from which attribute and method selections are made. Primitive types, types from java.lang package and supertypes are not counted.

Excessive coupling between objects is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be

**Ce** (*Coupling Efferent*)
Efferent coupling counts the number of types within the measured package which depend upon types outside that package (outgoing dependencies). A high *Ce* value indicates that the package's stability depends too much on the stability of other packages.

**CF** (*Coupling Factor*)
This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator represents the number of non-inheritance couplings. The denominator is the maximum possible number of couplings in a system.

**DIT** (*Depth of Inheritance Tree*)
Counts how far down the inheritance hierarchy a class is declared. High values imply that a class is quite specialized.

**H** (*Relational Cohesion*)
Relational cohesion represents the relationship that the package has to all its types. It is the average number of internal relationships per type of the measured package. If $N$ is the number of types within the package and $R$ the total number of relationships that are directed to types of this package, then: $H=(R+1)/N$.

**LCOM** (*Lack of Cohesion of Methods*)
Takes each pair of methods in the class and determines the set of fields they each access. If they have disjoint sets of field accesses, the count $P$ increases by one. If they share at least one field access, $Q$ increases by one. After considering each pair of methods: RESULT = $(P > Q)$? $(P - Q)$: 0

A low value indicates high coupling between methods, which indicates high testing effort because many methods can affect the same attributes. This also indicates potentially low reusability.

**LOC** (*Lines of Code*)
This is the traditional measure of size. It counts the number of code lines. Documentation and implementation comments as well as blank lines can be optionally interpreted as code. Documentation comments are Javadoc comments for Java, C++ and C#. Implementation comments are any other type of comments

**NMI** (*Number of Methods Inherited*)
Is calculated as the percentage of the number of non-redefined operations with regard to the number of operations inherited

**NOC** (*Number of Classes*)
Counts the number of classes
**NOCC** (*Number of Child Classes*)
Counts the number of classes that inherit from a particular class, meaning the number of classes in the inheritance tree that are located below the class. A non-zero value indicates that the particular class is being reused. However, the abstraction of the class may be poor if there are too many child classes. Also keep in mind that a high value for this measure points to the definite amount of testing required for each child class.
**NOP** (*Number of Polymorphic Methods*)
The metric is a count of the methods that exhibit polymorphic behavior. Such methods in C++ are marked as virtual.
**WMPC1** (*Weighted Method Per Class-1*)
This metric is the sum of the complexity of all methods for a class, where each method is weighted by its cyclomatic complexity. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class. Only methods specified in a class are included, that is, any methods inherited from a parent are excluded.
**WMPC2** (*Weighted Method Per Class-2*)
This metric is intended to measure the complexity of a class, assuming that a class with more methods than another is more complex, and that a method with more parameters than another is also likely to be more complex. The metric counts methods and parameters for a class. Only methods specified in a class are included, that is, any methods inherited from a parent are excluded.

## Appendix C. Papers merged into one study

Table C1

| Papers | Study | Description |
|---|---|---|
| P058 and P059 | P059 | The later paper build on S58 and adds an additional pattern matching algorithm to the method, i.e. automata simulation (bit-vector based). |
| P027 and P028 | P028 | Parsing language techniques. The later version includes some negative criteria that enhance the detection process. |
| P029, P030 and P031 | P029 | S30 introduces a two phase approach for identifying design pattern instances. S31 uses dynamic analysis in the second phase to enhance detection. S29 is the journal publication of the algorithm that is enhanced in terms of performance and accuracy. |
| P004, P005 and P006 | P004 | Similar logic and technology. Same year of publication. S5 is more based on metrics. S4 is a journal publication which additionally presents a tool and an evaluation on industrial code. |
| P014 and P015 | P014 | Same aim and method. The journal paper captures both behavioral and structural properties of GoF patterns and pattern variants. The conference paper only captures behavioral properties and doesnot deal with pattern variants. |
| P024, P025 and P026 | P026 | The first paper introduces the methodology, the second one evaluates it on GoF design patterns and the third one adds on how the designer can wait before the application of the pattern becomes beneficial in terms of flexibility |
| P009 and P033 | P033 | Same method (case study). Two out of three subjects are the same. Similar research questions, different level. One at pattern level, the other on pattern role level. |
| P012 and P013 | P013 | Same quality attribute, same method, different patterns |
| P042 and P064 | P042 | The later paper build on the previous one. The prior paper uses role based representation in UML. The journal paper specifies patterns through class and sequence diagrams |
| P069 and P070 | P070 | The later work is an extension of the previous one. Different goals |

## References

Alexander, C., Ishikawa, S., Silverstein, M., 1977. A Pattern Language: Town, Buildings, Construction. Oxford University Press, New York.
Ampatzoglou, A., Frantzeskou, G., Stamelos, I., 2012. A methodology to assess the impact of design patterns on software quality. Information and Software Technology, Elsevier 54 (April (4)), 331–346.
Ampatzoglou, A., Stamelos, I., 2010. Software engineering research for computer games: a systematic review. Information and Software Technology, Elsevier 52 (September (9)), 888–901.
Avgeriou, P., Zdun, U., 2005. Architectural patterns revisited – a pattern language. In: 10th European Conference on Pattern Languages of Programs (EuroPLOP'05), 6–10 July 2005, Irsee, Germany.
Basili, V., Caldiera, G., Rombach, H.D., 1994. Goal Question Metric Approach Encyclopedia of Software Engineering. John Wiley & Sons528–532.
Bjork, S., Lundgren, S., Holopainen, J., 2003. Game design patterns. In: Lecture Note of the Game Design track of Game Developers Conference, March 4–8, San Jose, CA, USA.
Bereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software, Elsevier 80 (April (4)), 571–583.
Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1996. Pattern-Oriented Software Architecture. Wiley, West Sussex, UK.
Cai, K.Y., Card, D., 2008. An analysis of topics in software engineering – 2006. Journal of Systems and Software, Elsevier 81 (June (6)), 1051–1058.
da Silva, F.Q.B., Santos, A.L.M., Soares, S.C.B., França, A.C.C., Monteiro, C.V.F., 2010. A critical appraisal of systematic reviews in software engineering from the perspective of the research questions asked in the reviews. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM' 10), ACM, Bolzano, Italy, 16–17 September 2010.
Dormey, G.R., 1995. A model for software product quality. IEEE Transactions on Software Engineering 21 (February (2)), 146–162.
Dyba, T., Dingsoyr, T., 2008. Empirical studies of agile software development: a systematic review. Information and Software Technology, Elsevier 50 (August (9–10)), 833–859.
Galster, M., Avgeriou, P., 2012. Qualitative analysis of the impact of SOA patterns on quality attributes. In: Proceedings of the 12th International Conference on Quality Software (QSIC 2012), IEEE Computer Society, Xi'an, China, 27–29 August 2012.
Gamma, E., Helms, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, Reading, MA.
Glass, R.L., Vessey, I., Ramesh, V., 2002. Research in software engineering: an analysis of the literature. Information and Software Technology, Elsevier 44 (June (8)), 491–506.
Hauge, O., Ayala, C., Conradi, R., 2010. Adoption of open source software in software-intensive organizations – a systematic literature review. Information and Software Technology, Elsevier 52 (November (11)), 1133–1154.
Heckman, S., Williams, L., 2011. A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, Elsevier 53 (April (4)), 363–387.
Höfer, A., Tichy, W.F., 2007. Status of Empirical Research in Software Engineering. Lecture Notes in Computer Science, vol. 4336. Springer, Dagstuhl Castle, Germany, pp. 10–19.
ISO9126, 1992. Information Technology — Software Product Evaluation — Quality Characteristics and Guidelines for Their Use. International Organisation for Standardization, Geneva.
Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering, Technical Report EBSE-2007-001, Keele University & Durham University Joint Report, Staffordshire, UK.
Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering – a systematic literature review. Information and Software Technology, Elsevier 51 (January (1)), 7–15.
Kitchenham, B.A., Budgen, D., Brereton, O.P., 2011. Using mapping studies as the basis for further research – a participant-observer case study. Information and Software Technology, Elsevier 53 (June (6)), 638–651.
Kitchenham, B., Pfleeger, S.L., 1996. Software quality: the elusive target. IEEE Software, IEEE 13 (January (1)), 12–21.
Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O.P., Turner, M., Niazi, M., Linkman, S., 2010. Systematic literature reviews in software engineering – a tertiary study. Information and Software Technology, Elsevier 52 (August (8)), 792–805.
Noblit, G.W., Hare, R.D., 1988. Meta-ethnography: Synthesizing Qualitative Studies. Sage, London.
Petticrew, M., Roberts, H., 2006. Systematic Reviews in the Social Sciences: a Practical Guide. Blackwell Publications, MA, USA.
Souza, J., Matwin, S., Japkowicz, N., 2002. Evaluating data mining models: a pattern language. In: 9th Conference on Pattern Language of Programs (PLOP'02), Monticello, Illinois, 8–12 September 2002.
Vokáč, M., Tichy, W., Sjøberg, D.I.K., Arisholm, E., Aldrin, M., 2004. 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 (September (3)), 149–195.

Walia, G.S., Carver, J.C., 2009. A systematic literature review to identify and classify software requirement errors. Information and Software Technology, Elsevier 51 (July (7)), 1087–1109.

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

Zhang, C., Budgen, D., 2012. What do we know about the effectiveness of software design patterns. IEEE Transactions on Software Engineering, IEEE 38 (September–October (5)), 1213–1231.

**Dr. Apostolos Ampatzoglou** is an Assistant Professor at the Computer Science Department of the University of Groningen, where he carries research and teaching in the areaof software engineering. He received a PhD in Software Engineering from the Department of Informatics, Aristotle University of Thessaloniki, Greece. His research interests include design patterns, software metrics and computer games.

**Sofia Charalampidou** is a Master Student in the Software Engineering program at Chalmers University of Technology. She holds a BSc in Information Technology from the Alexander Technological Institute of Thessaloniki. Her research interests include software design, software architecture, embedded systems and literature reviews. Formerly, she has been a member of the Software Engineering Group (SwEng) of the Aristotle University of Thessaloniki where she participated in research activities.

**Dr. Ioannis Stamelos** is an Associate Professor at the Department of Informatics of the Aristotle University of Thessaloniki, where he carries out research and teaching in the area of software engineering. He holds a diploma of Electrical Engineering (1983) and a PhD in Computer Science by the Aristotle University of Thessaloniki (1988). His current research interests are focused on open source software engineering, software project management and software education. He has published more than 100 articles in international journals and conferences. He is/was the scientific coordinator or principal investigator for his University in over 20 research and development projects in Information & Communication Technologies with funding from national and international organizations.