Maejo Int. J. Sci. Technol. **2018**, *12*(03), 251-271

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Review

Methods to measure the impact of design patterns on software maintainability

Aziz Nanthaamornphong* and Rattana Wetprasit

College of Computing, Prince of Songkla University, Phuket Campus, Vichit Songkram Road, Amphur Kathu, Phuket 83120, Thailand

* Corresponding author, e-mail: aziz.n@phuket.psu.ac.th

Received: 3 June 2017 / Accepted: 20 November 2018 / Published: 26 November 2018

Abstract: Design patterns are solutions that can be reused in frequent software design problems. Researchers and practitioners have made various claims regarding the benefits of design patterns, and these claims often concern the impact of design patterns on software maintainability. Unfortunately, the methods to measure these effects are not consistently applied. To provide empirical evidence concerning the measurement methods used to evaluate the impact of specific design patterns on software maintainability, we conducted a systematic mapping study that gathered all available empirical evidence on software maintainability. The review identified 30 primary studies from a collection of 2,832 search results. Researchers used both the case studies and controlled experiments to collect the evidence. The human-based controlled experiments tended to use students as participants, while the case studies tended to use a more realistic environment. Overall, the results indicated that the researchers used different definitions of maintainability and thus they used different measurement methods to assess the impact of design patterns. It is important to consider the use of a standard measurement method to consistently measure quality across different software systems. Due to the relative lack of empirical evidence, there is a need for further research concerning the impact of design patterns on software maintainability.

Keywords: design patterns, empirical study, software quality, systematic mapping study, software maintainability

INTRODUCTION

A design pattern is a general solution that can be used in similar situations to a common software design problem. Design patterns are best practices drawn from various sources, such as building software applications, developer experiences and empirical studies. In fact, design patterns were first presented by Alexander et al. [1] in their book *Pattern Language* in the context of building architecture. Subsequently, Gamma et al. [2] proposed approaches to applying architectural patterns to object-oriented software design. They proposed 23 design patterns called the gang of four (GoF) patterns, which were categorised into three groups, i.e. structural, behavioural and creational. The use of design patterns allows a developer to estimate the number of classes and methods necessary to implement a solution because each design pattern includes a name, the elements and the relationships among the elements. Design patterns assist a development team to gain a consistent understanding of the design. In this way, team members will more easily understand the intent of the software designers.

After the GoF patterns were published, several studies asserted the benefits of using design patterns [3]. Despite these benefits, developers face challenges when using design patterns. For example, the use of some design patterns is challenging and time consuming [4]. Thus, there is a learning curve for software designers to use design patterns effectively. In addition, in perspective of software quality, the use of design patterns does not consistently provide the benefits to a software system. For example, software designed for highly flexible usage can require a complex implementation that is challenging to maintain.

In software development, software maintenance is one of the most important software quality concerns [5]. Approximately 60% of software development lifecycle costs are spent on software maintenance [6, 7]. Similarly, previous studies have revealed that software maintenance could cause complicated and expensive tasks in the software life cycle [8, 9]. Since software maintenance is critical, software engineers should consider software maintainability as a high priority during software system development. Failing to address this concern can lead to an increase in the maintenance cost. Additionally, the impact of design patterns on maintainability has been reported more than any other software quality attributes [10]. However, those impacts have been measured differently. To the best of our knowledge, there is little consensus concerning the measurements of the impact of design patterns on software maintainability.

Therefore, there is a need to collect empirical evidence on the state of the methods used for measuring the impact of design patterns on software maintainability. In this study we conduct a systematic mapping study to gather all empirical evidence concerning the measurement methods that researchers use to evaluate the impact of design patterns on software maintainability. A systematic mapping study is designed to give an overview of a research area through classification and counting contributions in relation to the categories of that classification [11]. It involves searching the literature in order to know what topics have been covered in the literature and where the literature has been published. Hence a mapping study allows researchers to discover research gaps and trends. Our study attempts to provide a comprehensive review of the current state of knowledge of the measures by including only primary studies that evaluate the impact of the GoF design patterns. Rather than focusing on typical software maintainability measurements, we aim at measurement methods which researchers have used to examine the impact of design patterns existing in the software system. Additionally, we emphasised empirical studies that report how the design patterns have effects on software maintainability.

This study provides an input for researchers and practitioners making decisions to improve their current work or activities. Practitioners can use the results to assist in identifying which measurement methods are most appropriate for their situation. Researchers can use the results to identify gaps in the empirical research and to expand their empirical body of knowledge. Rather than focusing on all quality aspects (e.g. performance, usability and portability) which are too broad a collection, we conducted a systematic mapping study to better understand how researchers empirically evaluated the impact of design patterns on maintainability.

RELATED WORK

This section provides background information and related work on software maintainability. It also briefly discusses prior literature reviews of related topics.

Software Maintainability

Several studies have proposed various models or frameworks to increase software quality and define measurements, features, quality and related attributes and characteristics. Next, we briefly describe the ISO 25010-2011 [12] standard which was defined by International Organisation for Standardisation, currently the most influential quality standard in the software engineering community.

The ISO 25010-2011 standard defines maintainability as 'the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers,' where modifications may include alteration, reinstallation of changes and improvements or adjustment of the software under the changes in the requirements and environment. Additionally, maintainability includes the following sub-characteristics: analysability, modularity, modifiability, reusability and testability. Table 1 provides the descriptions of these sub-characteristics.

Table 1. Sub-characteristics of maintainability [12]

Sub-characteristic	Description
Modularity	'The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact to other components'
Reusability	'The degree to which an asset can be used in more than one system, or in building other assets'
Analysability	'The degree to which the software product can be diagnosed for deficiencies or cause of failures in the software, or for the parts to be modified to be identified'
Modifiability	'The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality'
Testability	'The degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and with which tests can be performed to determine whether those criteria have been met'

One of the most well-known measurement processes for software quality was proposed by the ISO/IEC JTC 1 Subcommittee 7 (SC7) – System and Software Engineering [13]. SC7 proposed a generic measurement model that assists in determining what must be specified for measurement planning, performance and evaluation. This model, published in the ISO/IEC 15939 standard [14], states that a specific attribute measure can be collected from a particular measurement method. A base measure is an attribute and its quantification method. A derived measure, which is a collection of two or more base measures, is then employed in a formula to create and build a particular derived measure. An analysis model uses these derived measures as an indicator. The indicator's value is then interpreted to describe the relationship between the information need and the indicator's value, which can produce an information product.

In addition to the standard measurement process, Riaz et al. [15] performed a systematic review to collect evidence on software maintainability prediction and metrics. Based on 14 primary studies, several researchers have proposed software metrics to measure or indicate software maintainability, but few studies have used a standard model to evaluate maintainability. However, this study was conducted under a different context compared to our study, in which the impact of design patterns on maintainability was not a criterion of selecting the primary studies. Additionally, the study focuses only on metrics-based maintainability.

Prior Literature Review

We identified three existing mapping studies of design patterns. Here, we briefly describe those studies and explain how our current study expands this prior work.

First, Zhang and Budgen [16] summarised 11 studies published between 1995 and 2009 that reported empirical evidence concerning the effectiveness of design patterns. The study focused on overall effectiveness rather than any specific quality characteristics. Their work included selected studies that used empirical methods for specifically evaluating the effects of design patterns on maintainability. Our current review includes a total of 30 studies published between 1997 and 2016 (including the ones cited by Zhang and Budgen [17-21]) that focused on empirically evaluating the impact of the GoF design patterns on maintainability.

Second, Ampatzoglou et al. [10] presented a systematic mapping study of about 118 primary studies to summarise the current research state on GoF design patterns. They classified the research of GoF design patterns into the following groups: 1) formal descriptions of design patterns, 2) design pattern discovery approaches, 3) design pattern applications, 4) effects of design patterns on software quality and 5) miscellaneous. The papers presented in their work included both those that used empirical methods and those that used other methods such as conceptual analysis/mathematical and descriptive methods. Additionally, this review did not focus specifically on maintainability. The search results show that the most active research topics are *design pattern discovery approaches* and *how the GoF patterns impact quality contributes*. One of the key findings of their work was the observation that researchers have studied maintainability more than any other quality attribute.

Third, Mayvan et al. [22] performed a recent systematic mapping study to present an overview of the research effort of design patterns. The authors studied 637 primary studies. The results showed that the topics of design patterns can be divided into six groups: 1) design pattern development, 2) design pattern usage, 3) design pattern mining, 4) quality evaluation, 5) design pattern specification and 6) miscellaneous issues. The authors also concluded that design pattern research was an active topic in recent decades and that the trend of publications in this field is increasing.

Rather than focusing on the effects of design patterns on software maintainability, which those studies have reported, we seek to gather evidence of researchers' understanding of maintainability and their measurement methods. An enriched understanding of software maintainability would help researchers and practitioners in developing better software systems.

Additionally, to understand the measurement methods would provide meaningful contributions to the software engineering research community.

RESEARCH METHODOLOGY

This research is based on the systematic mapping study presented by Petersen et al. [11]. We developed a protocol for the review by following their guidelines and methods as follows.

Research Questions

Although design patterns are a well-studied and important topic, there has been no extensive systematic mapping study to gather empirical evidence of measuring the impact of design patterns on maintainability. The research questions in our mapping study are as follows.

Question 1: How is software maintainability understood by researchers and practitioners? The goal of this research question is to better understand how researchers and practitioners define software maintainability and to see whether those definitions are consistent.

Question 2: What techniques/methods do researchers use for measuring the impact of design patterns on maintainability? We intend to determine the measurement methods that researchers and practitioners have used to report the impact of design patterns used in software systems.

Sources for Primary Studies

We searched the following databases to identify journals, conferences and workshop proceedings that were candidates for inclusion in the review: IEEExplore, ACM Digital Library, INSPEC, EI Compendex and SpringerLink. We did not use Google Scholar because it provides the same results as the search results from our mentioned databases [23].

Selection Criteria

We constructed a set of search strings to search for primary studies in the databases by taking the following steps: 1) build the keywords from the research questions; 2) identify different synonyms and spellings for each keyword; 3) apply the Boolean OR incorporate the keywords (if applicable); and 4) apply the Boolean AND link the keywords (if applicable). Our initial search terms are as follows:

(design pattern) AND (empirical OR experiment OR evaluation) AND (maintainability OR maintenance)

We formulated the search strings by adding synonyms of the search terms. Because the term *design pattern* is a common term, we added the keywords such as *Title* and *Abstract* to the search strings to limit the results because papers that focus on design patterns should encompass the term *design pattern* in either the title or the abstract. The final search strings are as follows:

(Abstract: 'design pattern' OR Abstract: 'design pattern' OR Title: 'design pattern' OR Title: 'design patterns') AND ('case stud' OR experiment OR empirical OR evaluat OR measur* OR survey OR examin* or compar*) AND (maintain* OR maintenance)*

We modified the search strings to follow the rules of each search engine. We used two criteria to test whether the results obtained using the search strings addressed the relevant research questions. First, we ensured that the search results included the known primary studies [16, 21, 24-32]. Second, we ensured that the search results included relevant references from those primary studies.

Table 2 describes the inclusion and exclusion criteria we used to ensure that all papers were closely related to the research questions. In this study we focus only on the 23 GoF design patterns because these patterns are well known in the software engineering community.

Table 2. Inclusion/exclusion criteria

Inclusion criteria (all required for inclusion)
1. Papers that study one or more of the 23 GoF design patterns
2. Papers that empirically evaluate the impact of design patterns on software maintainability and its sub-characteristics
3. Papers that were published in either peer-reviewed journals or conference/workshop proceedings
4. Papers that were published before 2017
Exclusion criteria (each sufficient for exclusion)
1. Papers that are based only on expert opinion or experience and lack objective empirical evidence
2. Studies that are unrelated to any research questions
3. Studies that are not in English
4. Editorials, keynotes, books, panel discussions and technical reports

We used the studied selection criteria to assess the papers returned by the search process. The first author analysed the titles of the papers and eliminated irrelevant papers. Then the first author reviewed the abstracts of the remaining papers and eliminated any that were irrelevant. Finally, the first author reviewed the full text of the remaining papers. At each step, the first author eliminated only the papers that were clearly irrelevant. To satisfy the exclusion/inclusion criteria and avoid bias, at each phase the second author independently reviewed approximately 80% of eliminated papers. This practice is consistent with other systematic literature studies [33, 34]. For the small number of papers on which the two authors disagreed, a brief discussion resolved the disagreement. We excluded papers only when both authors agreed.

Data Collection

We developed a data extraction form to ensure that we consistently gathered important information from the selected papers. This form allowed us to capture necessary information from the selected papers and to be definite about how each of them responded our research questions. The details of the data extraction form including data items and quality assessment questions can be found in Table 3. The first author read and created the extraction form for each selected paper. To mitigate the faults occurring during the data extraction process, the second author separately reviewed a random sample of 80% of the data extraction forms and then the authors compared their results in a face-to-face meeting. Whenever the data extracted were different (the differences being greater than 15%), the authors discussed these differences until an agreement was reached. We believe that the experience from a face-to-face meeting would reduce any bias for the remaining 20% of the primary studies.

Table 3. Data extraction form

Data Item	Description
Citation	Citation detail including authors, title, venue, date, publisher, etc.
Study objectives	The aims or goal of the primary study
Research Question 1 How is software maintainability understood by researchers and practitioners?	
The definition	Definitions of software maintainability or sub-characteristics included in the study
Research Question 2 What techniques/methods do researchers use for measuring the impact of design patterns on maintainability?	
Study type	Controlled experiment, case study, survey, others
Type of participants	Students, professionals, mixed (if applicable)
Number of participants	Number of participants in the study (if applicable)
Analysis method	Data analysis method that was used in the study, e.g. qualitative, quantitative, or mixed, any statistic methods
Measurement method	Detail of the methodology used to measure the effect of the design pattern(s)
System / application	Identify the name of the software or application
Application domain	Identify the domain of the application, e.g. financial, software development tool
Platform	How the software is executed, e.g. standalone, web application, mobile application
Open source	Yes/No
Notation	Notation used to describe the design patterns, e.g. UML
Programming language	e.g. C++, Java, C#
Design patterns	Which design pattern(s) were included in the study?
Major limitation	Main limitation of the study

REPORTING THE REVIEW

In this section we present the results for the questions posed in the previous section. We conducted the systematic mapping study according to the steps described in the research methodology section. The search strings identified 2,832 papers. A detailed title analysis reduced this number to 199. Reading the abstracts further reduced the number to 80. We reviewed the full-text of these 80 papers and chose 30 to include in the review. Table 4 presents the publication sources, publication kinds, number of papers and percentages of the total numbers of selected papers. The search resulted in papers from the main software engineering conferences and journals. Undoubtedly, conferences on software maintenance and empirical software engineering are at the top of the ranking because we included only empirical studies and focused on the maintainability of design patterns. Figure 1 shows the percentage of selected studies that have been published by each publication venue.

Table 4. Sources of selected papers

Source	Type	Count	%
International Workshop on Replication in Empirical Software Engineering Research	Proceedings	4	13.33
International Conference on Software Maintenance	Proceedings	3	10
International Symposium on Empirical Software Engineering and Measurement	Proceedings	3	10
Empirical Software Engineering	Journal	2	6.67
Transaction on Software Engineering	Journal	2	6.67
International Conference on Quality Software	Proceedings	1	3.33
International Conference on Model Driven Engineering Languages and Systems	Proceedings	1	3.33
International Symposium on Foundations of Software Engineering	Proceedings	1	3.33
Information and Software Technology	Journal	1	3.33
European Software Engineering Conference	Proceedings	1	3.33
International Symposium on Software Metrics	Proceedings	1	3.33
Computers and Education	Journal	1	3.33
Journal of Computer Science and Technology	Journal	1	3.33
International Working Conference on Reverse Engineering	Proceedings	1	3.33
International Conference on Objects, Models, Components, Patterns	Proceedings	1	3.33
Information Systems Development	Journal	1	3.33
International Conference on Software Engineering	Proceedings	1	3.33
International Workshop on Process Modelling and Empirical Studies of Software Evolution	Proceedings	1	3.33
International Workshop on Empirical Studies of Software Maintenance	Proceedings	1	3.33
Information Software and Technology	Journal	1	3.33
International Conference on Applications for Software Engineering, Disaster Recovery and Business Continuity	Proceedings	1	3.33
Total		30	100.00

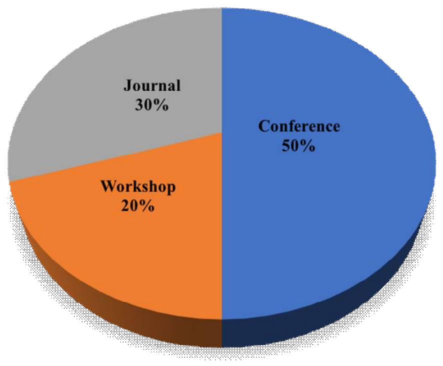

Figure 1. Number of publications across publication type

Research Question 1: How is software maintainability understood by researchers and practitioners?

Overall, only 4 of the 30 studies provided an explicit definition of maintainability (Table 5). Of the definitions provided by those four studies, however, only one study [35] mentioned a quality model (ISO 14764) as the basis of maintainability. The ISO 14764 standard guides the planning, execution and control, review and assessment, and closure of the software maintenance process. The scope of the standard involves maintenance of multiple software systems with the same maintenance means. The researchers measure maintainability in terms of the maintenance scenario categorised according to the ISO 14764 standard, including perfective, corrective and adaptive. Scanniello et al. [36] cited the definition of a maintenance process from Swanson [37] in their work. Swanson identified three types of changes that can be performed during software development: 1) corrective, 2) perfective and 3) adaptive. A corrective maintenance task is a reactive modification to correct identified problems, while a perfective maintenance task is required to improve the quality of an existing software system. An adaptive task is to maintain a software system's usability in a changed or evolving environment.

Table 5. Definitions of maintainability

Study	Definition
Hills et al. [35]	The researchers follow the definition of maintainability from ISO 14764. Maintainability includes perfective, corrective and adaptive maintenance. *Perfective* is 'the modification of a software product after delivery to detect and correct latent faults in the software products before they are manifested as failures.' *Corrective* is 'the reactive modification of a software product performed after delivery to correct discovered problems.' *Adaptive* is 'the modification of a software product, performed after delivery, to keep a software product usable in a changed or changing environment.'
Scanniello et al. [36]	To make a software system continuously meet users' satisfaction, the system requires a maintenance process including 1) corrective, 2) perfective and 3) adaptive.
Posnett et al. [38]	More change-proneness of classes in design patterns tends to require more maintenance effort.
Schanz and Isurieta [39]	Modular grime is defined as increase in the internal and external coupling of classes that belong to a pattern.

In summary, only four studies explicitly defined maintainability. Furthermore, those definitions were inconsistent and did not conform with the ISO 25010-2011 standard.

Research Question 2: What techniques/methods do researchers use for measuring the impact of design patterns on maintainability?

To better interpret the results of the studies, it is important to know what type of study provided the data. We identified two primary study types: 1) controlled experiments and 2) case studies. The details of each study type are as follows.

Controlled experiments

A controlled experiment investigates a hypothesis by handling one or more independent variables (e.g. a particular design pattern) to measure their impacts on the dependent variable (e.g. comprehension or correctness). Each combination of values for the independent variable(s) is a treatment [40].

In this review the majority of the controlled experiments used human participants as opposed to software systems or other units of analysis. Fourteen studies used undergraduate or graduate students [17-19, 26-30, 36, 41-45]. Only two studies used professional participants [31, 46] and one used both students and professional participants [21]. The other two controlled experiments [24, 47] were not human-based.

Regarding the materials used in the human-based and non-human-based experiments, those studies used open source, self-developed and commercial software. The most common open source software used was JHotDraw (http://www.jhotdraw.org). Table 6 summarises each of the 19 controlled experiments along with the study objective or research questions and the software or materials used.

Table 6. Controlled experiments

Study	Objectives/Research questions	Participants/Software materials
Jeanmart et al. [17]	To study whether the Visitor pattern has an impact on modification and comprehension tasks based on UML class diagrams	Postgraduate; graduate students / JHotDraw; JRefactory; Ptidej (design patterns vs non-pattern)
Ng et al. [18]	To investigate how practitioners use design patterns and, when they do, which problems they frequently address	215 Undergraduate students / JHotDraw; Multiple-user Calendar Manager (MCM); Hotel Management System (HMS) (design patterns vs non-pattern)
Prechelt et al. [19]	To assess whether the use of design patterns with their documentation can support software practitioners	64 Graduate students; 32 undergraduate students / And/Or-tree; Phonebook (design patterns vs non-pattern)
Vokáč et al. 2004 [21]	To compare the maintainability of 2 program versions (design patterns vs non-design pattern)	39 Professional engineers; 5 graduate students / Boolean Formulas (BO); Communication Library (CO); Graphics Library (GR); Stock Ticker (ST) (design patterns vs. non-pattern)
Bieman et al. [24]	To study the maintenance tasks when the program structures are changed	Netbeans; Java Java Commercial Systems; JRefactory; C++ Commercial System (non-human-based experiment)
Garzás et al. [26]	To examine whether design patterns enhance the understandability of software designs	10 Undergraduate students; 37 graduate students / aeronautics; bank; university; newspaper agency (design patterns vs non-pattern)
Juristo and Vegas [27]	To evaluate whether the design patterns have an effect on software maintainability	8 Graduate students / communication on library; graphics library (design patterns vs non-pattern)
Krien at al. [28]	To evaluate whether the design patterns have an effect on software maintainability	Students / communication on library; graphics library (design patterns vs non-pattern)

Table 6. (continued)

Nanthaamornphong and Carver [29]	To evaluate whether the design patterns have an effect on software maintainability	18 Graduate students / Boolean formulas; communication on library; graphics library; stock ticker (design patterns vs non-pattern)
Prechelt and Liesenberg [30]	To evaluate whether the design patterns have an effect on software maintainability	13 Graduate students / communication on library; graphics library (design patterns vs non-pattern)
Prechelt et al. [31]	To evaluate whether the design patterns have an effect on software maintainability	29 Professional software engineers / Boolean formulas; communication on library; graphics library; stock ticker (design patterns vs non-pattern)
Scanniello et al. [36]	To assess the effort and efficiency of the practitioner performing maintenance tasks on the source code that design patterns are documented with.	24 Graduate students / JHotDraw version 5.1 (design patterns vs non-pattern)
Chatzigeorgiou et al. [41]	To assess the ability of students to understand design patterns based on an assignment	74 Undergraduate students
Gravino et al. [42]	To investigate whether the design pattern with the document improves the maintainability	24 Graduate students; 17 graduate students / JHotdraw (design patterns vs non-pattern)
Ng et al. [43]	To study whether the refactoring by using design patterns can support a maintenance task	55 Undergraduate students; 63 graduate students / JHotDraw (design patterns vs non-pattern)
Ng et al. [44]	To investigate how to use design patterns effectively	118 Students / JHotDraw (design patterns vs non-pattern)
Prechelt et al. [45]	To assess whether the use of design patterns with their documentation can support software practitioners	74 Graduate students / Element; Tuple (design patterns vs non-pattern)
Gravino et al. [46]	To evaluate whether the presence of design pattern instances affects source code understanding	25 Professional developers / JHotDraw (design patterns vs non-pattern)
Aversano et al. [47]	To study the maintainability of evolved open source applications including design patterns	Eclipse-JDT; JHotDraw; ArgoUML. (non-human-based experiment)

Case studies

A case study allows a researcher closely examines an interesting issue within a specific context [48]. In software engineering fields researchers typically use the case study approach when studying an existing software system. For example, Ampatzoglou and Chatzigeorgiou [49] investigated the benefits of using design patterns in game development. They analysed both qualitative and quantitative data collected from a study. Table 7 summarises the 11 case studies along with the objective or research questions and the software or materials used.

Table 7. Case studies

Study	Objectives/Research questions	Software/Materials
Tichy and Unger [20]	To investigate whether the use of design patterns can help the communication in the team	ChiCo; Timmie
Penta et al. [25]	To examine whether design pattern roles have more change-prone than others	JHotDraw; Xerces-J; Eclipse-JDT
Hills et al. [35]	To compare two design patterns (Visitor vs Interpreter)	Add; Gen; Resolve; Lambda
Posnett et al. [38]	To investigate whether classes playing the roles in design patterns have trend to change	JHotDraw; Xerces; EclipseJDT
Schanz and Isurieta [39]	To investigate the relationship between modular grime and design patterns	Vuze (formally Azuereus)
Ampatzoglou and Chatzigeorgiou [49]	To investigate the use of design patterns in game development	Cannon smash; Ice hockey manager
Bieman et al. [50]	To study the maintenance tasks when the program structures are changed	A commercial OO application (medium size)
Jovičić and Vlajić [51]	To improve the enterprise resource planning system by using design patterns	Dynamic AX; Dynamics GP Dynamics NAV; SAP; Oracle EBS
Khomh et al. [52]	To study the impact of classes playing the role(s) in design patterns on software quality	ArgoUML v.0.18.1; Azureus v.2.1.0.0; JDT Core v.2.1.2; Xalan v.2.7.0; JHotDraw v5.4b2; Xerces v.1.4.4
Ampatzoglou et al. [53]	To propose an analytical method to assess how design patterns have an effect on software quality	Open source applications (authors do not provide the name.)
Hegedűs et al. [54]	To evaluate whether the design patterns have an effect on software maintainability	JHotDraw v.7.0

Categories of measurement methods

Based on these two study types, we classify the measurement methods that researchers used into four categories as follows.

1) **Efficiency of maintenance tasks**: These studies measure maintainability through correctness and time to complete the given maintenance tasks [17, 20, 21, 26-31, 43, 44].
2) **Change-proneness**: These studies measure maintainability as the frequency of changes in classes participating in the design patterns [18, 24, 25, 38, 47, 50].
3) **Design pattern documentation**: These studies analyse whether documentation of design patterns affect the efficiency (measured by correctness and time) of answering survey questions regarding maintenance tasks [19, 36, 42, 45, 46].
4) **Software metrics**: These studies measure maintainability through software metrics, for example cyclomatic complexity, coupling factors and lack of cohesion methods [35, 39, 41, 49, 51-54].

Figure 2 shows the percentages of studies that used these methods. The *efficiency of maintenance tasks* is the method most employed by researchers in their studies (11 studies or 36%), followed by software changes, software metrics and design pattern documentation. In general, researchers examined the maintainability by comparing the software developed in two different versions (design patterns vs non-design pattern). Only one study [35] compared specific design against each other. Table 8 briefly describes the measurement methods.

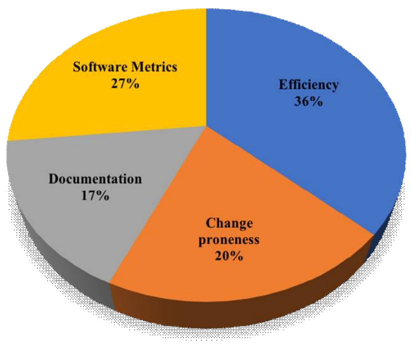

Figure 2. Percentages of measurement methods

Table 8. Measurement methods

Software Metric	Purpose
Complexity	
Attribute complexity [49]	The attribute complexity metric is defined as the sum of each attribute's value in the class (each type and array type have a predefined complexity value), so the complexity is increasing while the value is increasing.
Cyclomatic complexity [51]	Complexity is determined by the number of decision points in a method plus one for the method entry.
Maintenance complexity [35]	Maintenance complexity is defined as an aggregate sum of weighted occurrences of uses of programming language constructs.
Weighted methods per class [41, 49, 52]	It is simply the number of methods defined or implemented within a class. It measures the complexity of an individual class.
Coupling and Cohesion	
Coupling factor [49]	It is calculated as a fraction of which the numerator represents the number of non-inheritance couplings and the denominator is the maximum possible number of couplings in a system.
Lack of cohesion of methods [49, 41, 52]	It measures if a class of the system has all its methods working together in order to achieve a single, well-defined purpose.
Strength of coupling [39]	Strength is determined by the difficulty of removing the coupling relationship.
Scope of coupling [39]	Scope demarcates the boundary of a coupling relationship and can be internal or external.
Direction of coupling [39]	This metric uses afferent (Ca) and efferent (Ce) coupling to refer to the direction of a coupling relationship.
'C' connectivity of a class [52]	Hitz and Montazeri [55] restate the definition of lack of cohesion of methods based on graph theory. Lack of cohesion of methods is defined as the number of connected components of a graph.
Coupling between objects [41, 52]	It represents the number of classes coupled to a given class.
Number of classes inherited [52]	Number of derived classes for a class.
Number of class inheriting [52]	Number of inheriting classes for a class.
Number of methods overridden by a class [52]	Number of overridden methods for a class.

Table 8. (continued)

Depth of inheritance of tree in class [52]	It provides for each class a measure of the inheritance levels from the object hierarchy top.
Size	
Lines of code [41]	Number of lines of program code
Number of children [41]	It measures the number of immediate descendants of the class.
Number of attributes [41]	It measures the average number of attributes for a class.
Number of operations [41, 52]	Number of methods or operations in a class

DISCUSSION

This section discusses the principal findings of this study, suggestions for practitioners and researchers, and the limitations of this study.

Principal Findings

It was found in this investigation that researchers did not always provide a clear definition of maintainability. Several studies provided a definition of maintainability, but those definitions were not consistent. From 30 selected studies, only one explicitly provided the definition corresponding to a quality model standard such as ISO/IEC or IEEE standard (IEEE Std. 610.12-1990 [56]). Subsequently, there has been no single measure for the application's software maintainability and it is necessary to measure many characteristics based on given definitions. We could see the various methods that researchers have used to measure the impact of design patterns on software maintainability. We believe that the use of different methods was one of the reasons that the software engineering research community could not summarise how design patterns have effects on maintainability. To ensure that the impact of design pattern on maintainability is measured in the same way, software engineering researchers should use the standard definition of maintainability because if they understand *what* maintainability is, then the question of *how* to measure maintainability can be consistently answered. Similarly, the impact of design pattern on other quality attributes (e.g. performance, reusability) should be measured along with the consistent definitions.

With respect to software metrics, the results suggest that 27% of all primary studies used software metrics to quantify software maintainability [35, 39, 41, 49-51]. These results show that software metrics were not employed as much as we expected even though the software metrics were designed to quantify the quality of software. Additionally, many software maintainability metrics are available [15]. The results also show that most studies did not consider a specific type of maintenance (e.g. corrective or perfective) when discussing maintainability. Only two studies [43, 44] considered perfective maintenance. Additionally, the results show that complexity was most commonly used to measure maintainability. Rather than using the code complexity metric directly, for example, Hills et al. [35] used maintenance complexity to evaluate maintainability. The researchers designed several maintenance scenarios and calculated the maintenance complexity. For each maintenance scenario, the researchers compared the computed complexity of a designed activity required to perform the maintenance scenario for two versions (e.g. versions 1 and 2).

We found that researchers combined various metrics to measure the effect of design patterns on maintainability. For example, Chatzigeorgiou et al. [41] used a set of software metrics in their study. They used metrics including number of attributes, lines of code, number of operations and number of classes to evaluate the increased size in many cases when the system included design patterns. They also used code complexity, lack of cohesion in methods (LCOM), weighted methods per class (WMC) and coupling between objects (CBO) to evaluate the maintenance difficulties. They stated that code complexity reflects how easy the code is to understand and modify. Similarly, they used CBO, LCOM and WMC to determine the ripple effect of the system being modified.

Table 8 summarises the software metrics that we discover in the literature. We classify those metrics into three categories based on their objectives, i.e. 1) complexity, 2) coupling and cohesion and 3) size. The first column lists the software metrics employed in the selected studies and in the second column their purposes or definitions are described. The results show that researchers commonly used software complexity and coupling-based metrics to measure the maintainability. Complexity metrics can be useful predictors of the maintenance behaviour of systems and we recommend a greater use of measurements during the system development and maintenance. This evidence highlights the fact that there are few maintainability metrics and predictors, particularly software metrics that are related to the effect of design patterns. Thus, there is an opportunity for innovating new maintainability prediction metrics.

The literature review identifies two types of studies that were used to investigate the impact of design patterns on software maintainability: *controlled experiments* and *case studies*. Each type of study has its own goals and contributions. The complementary nature of the approaches of these two studies suggests that researchers should use the findings from controlled experiments and attempt to replicate them in the case studies and that they should use results from the case studies and attempt to better understand them via controlled experiments. Tables 6 and 7 summarise the controlled experiments and case studies respectively.

Regarding the *controlled experiments*, there were two primary axes over which they varied: whether human participants were used and what design patterns they were compared against. First, we found experiments that focused on human participants and those that did not focus on human participants. For the former case, the vast majority used student participants as opposed to professional developers. While there can be some benefits of conducting experiments with students as participants [57, 58], there is also a need to replicate many of these studies with professional developers to see whether the results are reliable in a more realistic setting. Second, the majority of the controlled experiments compared the effects of using design patterns to those of not using design patterns. This void in the literature suggests a fruitful research opportunity to compare design patterns and to study the outcomes when multiple design patterns are used together.

Conversely, all *case studies* were conducted in a realistic environment, typically on commercial software. Although the majority of the case studies did use commercial software, only a few of them were of large commercial systems. In terms of programming language, the majority of these systems were written in either the Java or C++ programming languages. Researchers may use the results of the case studies to conduct controlled experiments to validate the results in different environments. However, they could encounter problems if the commercial system (or an equivalent) is not accessible. Based on the literature, we did not find any controlled experiment that replicates the case study.

Figure 3 presents the relationship between the measurement method and study type across the studies. We found that the efficiency of maintenance tasks was primarily used to measure the effect of design patterns in the controlled experiments. Conversely, software metrics were primarily used in the case studies. The majority of the measurement methods were used in both the controlled experiments and the case studies, with the exception of the pattern documentation method, which was used only in the controlled experiments.

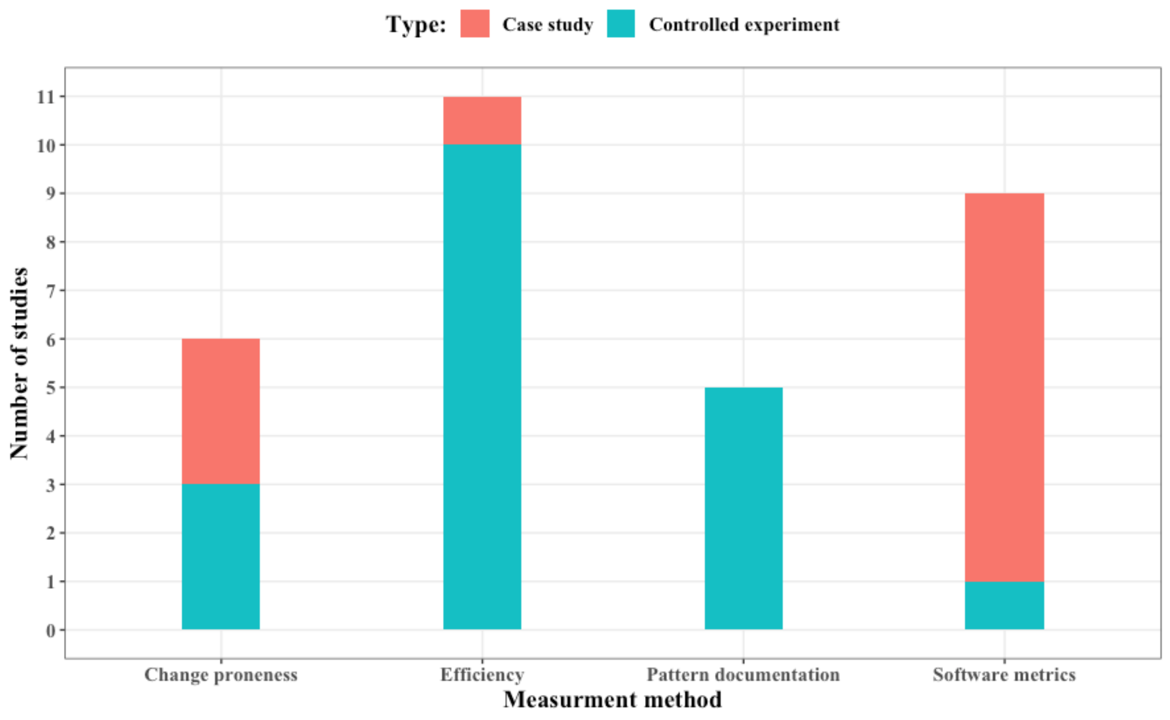

Figure 3. Relationship between measurement method and study type

Other types of empirical studies often employed in software engineering, e.g. survey and ethnography, were lacking in the literature. We did locate one paper describing a survey [59] and one based on expert opinion, but we excluded those papers because they were based only on opinion (one of our exclusion criteria).

Limitation*s*

Throughout the study we checked whether the extracted data could provide answers to the research questions. We refined certain data fields in the data extraction form to ensure that all data items conformed to the research goals. The potential shortcoming is selection bias, in which the correct set of studies is not chosen. To address this possibility, we defined inclusion and exclusion criteria to provide an assessment of how the final set of primary studies was obtained.

It is important to note, however, that the abstract of software engineering papers might not be well-written. As a result, there is a possibility that relevant studies were omitted. To mitigate this risk, we monitored every step of the review process, particularly the search strategy. Furthermore, since we focused on *empirical evidence*, we excluded 'lessons learned' and 'expert opinion' papers.

In comparing the results of this mapping study against the findings from three previous studies [10, 16, 22], we found many similarities. Therefore, the results of our study provide some support for generalising the results of the entire set of studies. Our study focuses only on the 23 GoF design patterns; thus, the results of this study may not be generalisable to other design patterns.

CONCLUSIONS

The main contribution of this systematic mapping study is the gathering and report of all existing empirical evidence of the impact that various design patterns have on software maintainability. The results of the impact of design patterns on software maintainability were collected from both the controlled experiments and case studies. Based on the evidence gathered from the study, we can conclude that in general, software maintainability was not clearly defined by the researchers. Thus, we adopt different methods for measuring the impact of design patterns. Based on the evidence gathered, we can provide conclusions and recommendations for practitioners and researchers.

This literature review highlights the need for additional empirical evaluation of other software quality aspects. Specific recommendations for researchers include the following:

- Provide practitioners with suitable metrics for assessing the effects of design patterns in their software.
- Increase the number of studies and replications performed with professional developers in the context of real systems to help practitioners evaluate the practical usefulness of the results.

From the practitioner’s perspective, this review indicates the importance of carefully considering the impact of each design pattern on software maintainability. More evidence will enable the practitioners to make better decisions on software maintainability in the future. Specific recommendations for practitioners include the following:

- Documentation of design patterns is important and source code comments should detail the deployed pattern to support maintenance.
- Software metrics may be the first choice for evaluating the effects of design patterns on the software system.

REFERENCES

1. C. Alexander, S. Ishikawa and M. Silverstein, "A Pattern Language: Towns, Buildings, Construction", Oxford University Press, New York, **1977**, pp.1-2
2. E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-oriented Software", Addison-Wesley Professional, Boston, **1994**, pp.4-5
3. L. Aversano, L. Cerulo and M. D. Penta, "Relating the evolution of design patterns and crosscutting concerns", Proceedings of 7th IEEE International Working Conference on Source Code Analysis and Manipulation, **2007**, Paris, France, pp.180-192.
4. M. P. Cline, "The pros and cons of adopting and applying design patterns in the real world", *Commun. ACM*, **1996**, *39*, 47-49.
5. D. Coleman, D. Ash, B. Lowther and P. Oman, "Using metrics to evaluate software system maintainability", *Computer*, **1994**, *27*, 44-49.
6. S. W. L. Yip and T. Lam, "A software maintenance survey", Proceedings of 1st Asia-Pacific Software Engineering Conference, **1994**, Tokyo, Japan, pp.70-79.
7. M. V. Zelkowitz, "Perspectives on software engineering", *ACM Comput. Surv.*, **1978**, *10*, 197-216.
8. M. J. C. Sousa and H. M. Moreira, "A survey on the software maintenance process", Proceedings of International Conference on Software Maintenance, **1998**, Bethesda (MD), USA, pp.265-274.

9. J. C. Chen and S. J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability", *J. Syst. Softw.*, **2009**, *82,* 981-992.
10. A. Ampatzoglou, S. Charalampidou and I. Stamelos, "Research state of the art on GoF design patterns: A mapping study", *J. Syst. Softw.*, **2013**, *86*, 1945-1964.
11. K. Petersen, S. Vakkalanka and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update", *Inf. Softw. Technol.*, **2015**, *64*, 1-18.
12. ISO/IEC, "ISO/IEC 25010 (2011) - Systems and software quality requirements and evaluation - System and software quality models", International Organization for Standardization, Geneva, **2011**.
13. ISO/IEC, " ISO/IEC JTC 1/SC 7 Software and systems engineering, information technology – software product quality – part 1: Quality model", International Organization for Standardization, Geneva, **1987**.
14. ISO/IEC, "ISO/IEC 15939: 2017 Systems and software engineering – Measurement process", International Organization For Standardization, Geneva, **2007**.
15. M. Riaz, E. Mendes and E. Tempero, "A systematic review of software maintainability prediction and metrics", Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, **2009**, Lake Buena Vista (FL), USA, pp.367-377.
16. C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?", *IEEE Trans. Softw. Eng.*, **2012**, *38*, 1213-1231.
17. S. Jeanmart, Y. G. Gueheneuc, H. Sahraoui and N. Habra, "Impact of the visitor pattern on program comprehension and maintenance", Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, **2009**, Lake Buena Vista (FL), USA, pp.69-78.
18. T. H. Ng, S. C. Cheung, W. K. Chan and Y. T. Yu, "Do maintainers utilize deployed design patterns effectively?", Proceedings of 29th International Conference on Software Engineering, **2007**, Minneapolis (MN), USA, pp.168-177.
19. L. Prechelt, B. Unger-Lamprecht, M. Philippsen and W. F. Tichy, "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance", *IEEE Trans. Softw. Eng.*, **2002**, *28*, 595-606.
20. B. Unger and W. F. Tichy, "Do design patterns improve communication? An experiment with pair design", Proceedings of International Workshop Empirical Studies of Software Maintenance, **2000**, San Jose (CA), USA, pp.1-5
21. 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", *Empir. Softw. Eng.*, **2004**, *9*, 149-195.
22. B. B. Mayvan, A. Rasoolzadegan and Z. G. Yazdi, "The state of the art on design patterns: A systematic mapping of the literature", *J. Syst. Softw.*, **2017**, *125*, 93-118.
23. L. Chen, M. A. Babar and H. Zhang, "Towards an evidence-based understanding of electronic data sources", Proceedings of 14th International Conference on Evaluation and Assessment in Software Engineering, **2010**, Swinton, UK, pp.135-138.
24. 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 9th IEEE International Software Metrics Symposium, **2003**, Sydney, Australia, pp.40-49.

25. M. D. Penta, L. Cerulo, Y.-G. Gueheneuc and G. Antoniol, "An empirical study of the relationships between design pattern roles and class change proneness", Proceedings of IEEE International Conference on Software Maintenance, **2008**, Beijing, China, pp.217-226.
26. J. Garzás, F. García and M. Piattini, "Do rules and patterns affect design maintainability?", *J. Comput. Sci. Technol.*, **2009**, *24*, 262-272.
27. N. Juristo and S. Vegas, "Design patterns in software maintenance: An experiment replication at UPM – experiences with the RESER'11 joint replication project", Proceedings of 2nd International Workshop on Replication in Empirical Software Engineering Research, **2011**, Banff, Canada, pp.7-14.
28. J. L. Krein, L. J. Pratt, A. B. Swenson, A. C. MacLean, C. D. Knutson and D. L. Eggett, "Design patterns in software maintenance: An experiment replication at brigham young university", Proceedings of 2nd International Workshop on Replication in Empirical Software Engineering Research, **2011**, Banff, Canada, pp.25-34.
29. A. Nanthaamornphong and J. C. Carver, "Design patterns in software maintenance: An experiment replication at university of Alabama", Proceedings of 2nd International Workshop on Replication in Empirical Software Engineering Research, **2011**, Banff, Canada, pp.15-24.
30. L. Prechelt and M. Liesenberg, "Design patterns in software maintenance: An experiment replication at Freie university Berlin", Proceedings of 2nd International Workshop on Replication in Empirical Software Engineering Research, **2011**, Banff, Canada, pp.1-6.
31. L. Prechelt, B. Unger, W. F. Tichy, P. Brossler and L. G. Votta, "A controlled experiment in maintenance comparing design patterns to simpler solutions", *IEEE Trans. Softw. Eng.*, **2001**, *27,* 1134-1144.
32. M. Vokáč, "Defect frequency and design patterns: An empirical study of industrial code", *IEEE Trans. Softw. Eng.*, **2004**, *30*, 904-917.
33. B. Kitchenham, O. P. Brereton, D. Budgen, M. Tunner, J. Bailey and S. Linkman, "Systematic literature reviews in software engineering – a systematic literature review", *Inf. Softw. Technol.*, **2009**, *51*, 7-15
34. J. C. Carver, E. Hassler, E. Hernandes and N. A. Kraft, "Identifying barriers to the systematic literature review process", Proceedings of ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, **2013**, Baltimore (MD), USA, pp.203-213.
35. M. Hills, P. Klint, T. van der Storm and J. Vinju, "A case of visitor versus interpreter pattern", Proceedings of 49th International Conference on OBJECTS, Models, Components, Patterns, **2011**, Zurich, Switzerland, pp.228-243.
36. 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 4th International Symposium on Empirical Software Engineering and Measurement, **2010**, Bolzano-Bozen, Italy, pp.1-4.
37. E. B. Swanson, "The dimensions of maintenance", Proceedings of 2nd International Conference on Software Engineering, **1976**, San Francisco, USA, pp.492-497.
38. D. Posnett, C. Bird and P. Dévanbu, "An empirical study on the influence of pattern roles on change-proneness", *Empir. Softw. Eng.*, **2011**, *16*, 396-423.
39. T. Schanz and C. Izurieta, "Object oriented design pattern decay: A taxonomy", Proceedings of IEEE International Symposium on Empirical Software Engineering and Measurement, **2010**, Bolzano-Bozen, Italy, pp.1-8.

40. S. Easterbrook, J. Singer, M.-A. Storey and D. Damian, "Selecting empirical methods for software engineering research", in "Guide to Advanced Empirical Software Engineering" (Ed. F. Shull, J. Singer and D. I. K. Sjøberg), Springer, London, **2008**, pp.285-311.
41. A. Chatzigeorgiou, N. Tsantalis and I. Deligiannis, "An empirical study on students' ability to comprehend design patterns", *Comput. Educ.*, **2008**, *51*, 1007-1016.
42. C. Gravino, M. Risi, G. Scanniello and G. Tortora, "Does the documentation of design pattern instances impact on source code comprehension? Results from two controlled experiments", Proceedings of 18th Working Conference on Reverse Engineering, **2011**, Limerick, Ireland, pp. 67-76.
43. 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 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, **2006**, Portland (OR), USA, pp.12-22.
44. T. H. Ng, Y. T. Yu and S. C. Cheung, "Factors for effective use of deployed design patterns", Proceedings of 10th International Conference on Quality Software, **2010**, Zhangjiajie, China, pp.112-121.
45. L. Prechelt, B. Unger and M. Philippsen, "Documenting design patterns in code eases program maintenance", Proceedings of ICSE Workshop on Process Modeling and Empirical Studies of Software Evolution, **1997**, Boston (MA), USA, pp.72–76.
46. C. Gravino, M. Risi, G. Scanniello and G. Tortora, "Do professional developers benefit from design pattern documentation? A replication in the context of source code comprehension", Proceedings of 15th International Conference on Model Driven Engineering Languages and Systems, **2012**, Innsbruck, Austria, pp.185-201.
47. L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso and M. D. Penta, "An empirical study on the evolution of design patterns", Proceedings of 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, **2007**, Dubrovnik, Croatia, pp.385-394.
48. R. K. Yin, "Case Study Research: Design and Methods", 3rd Edn., Sage Publications, Thousand Oaks (CA), **2002**, pp.385-394.
49. A. Ampatzoglou and A. Chatzigeorgiou, "Evaluation of object-oriented design patterns in game development", *Inform. Softw. Technol.*, **2007**, *49*, 445-454.
50. J. M. Bieman, D. Jain and H. J. Yang, "OO design patterns, design structure, and program changes: An industrial case study", Proceedings of IEEE International Conference on Software Maintenance, **2001**, Florence, Italy, pp.580-589.
51. B. Jovičić and S. Vlajić, "Design patterns application in the ERP systems improvements", in "Information Systems Development: Towards a Service Provision Society" (Ed. G. A. Papadopoulos, W. Wojtkowski, G. Wojtkowski, S. Wrycza and J. Zupancic), Springer, Boston, **2010**, pp.451-459.
52. F. Khomh, Y.-G. Gueheneuc and G. Antoniol, "Playing roles in design patterns: An empirical descriptive and analytic study", Proceedings of IEEE International Conference on Software Maintenance, **2009**, Edmonton, Canada, pp.83-92.
53. A. Ampatzoglou, G. Frantzeskou and I. Stamelos, "A methodology to assess the impact of design patterns on software quality", *Inf. Softw. Technol.*, **2012**, *54*, 331-346.
54. P. Hegedűs, D. Bán, R. Ferenc and T. Gyimóthy, "Myth or reality? Analyzing the effect of design patterns on software maintainability", in "Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity: International Conferences" (Ed. T.-

H. Kim, C. Ramos, H.-K. Kim, A. Kiumi, S. Mohammed and D. Ślęzak), Springer, Berlin, **2012**, pp.138-145.

55. M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented system", Proceedings of Interntional Symposium on Applied Corporate Computing, **1995**, Monterrey, Mexico, pp.1-9.
56. IEEE, "IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)", Computer Society of the IEEE, New York, **1990**.
57. J. Carver, L. Jaccheri, S. Morasca and F. Shull, "Issues in using students in empirical studies in software engineering education", Proceedings of 9th IEEE International Software Metrics Symposium, **2003**, Sydney, Australia, pp.239-249.
58. J. C. Carver, L. Jaccheri, S. Morasca and F. Shull, "A checklist for integrating student empirical studies with research and teaching goals", *Empir. Softw. Eng.*, **2010**, *15*, 35-59.
59. F. Khomh and Y.-G. Gueheneuce, "Do design patterns impact software quality positively?", Proceedings of 12th European Conference on Software Maintenance and Reengineering, **2008**, Athens, Greece, pp.274-278.