Information and Software Technology 65 (2015) 14–38

Contents lists available at ScienceDirect

Information and Software Technology

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

How have we evaluated software pattern application? A systematic mapping study of research design practices

Maria Riaz [a,*], Travis Breaux [b], Laurie Williams [a]

[a] North Carolina State University, Dept. of Computer Science, Raleigh, NC, USA
[b] Carnegie Mellon University, Institute for Software Research, Pittsburgh, PA, USA

ARTICLE INFO

Article history:
Received 7 October 2014
Received in revised form 2 April 2015
Accepted 2 April 2015
Available online 9 April 2015

Keywords:
Software pattern
Mapping study
Systematic review
Empirical evaluation
Empirical design

ABSTRACT

Context: Software patterns encapsulate expert knowledge for constructing successful solutions to recurring problems. Although a large collection of software patterns is available in literature, empirical evidence on how well various patterns help in problem solving is limited and inconclusive. The context of these empirical findings is also not well understood, limiting applicability and generalizability of the findings.
Objective: To characterize the research design of empirical studies exploring software pattern application involving human participants.
Method: We conducted a systematic mapping study to identify and analyze 30 primary empirical studies on software pattern application, including 24 original studies and 6 replications. We characterize the research design in terms of the questions researchers have explored and the context of empirical research efforts. We also classify the studies in terms of measures used for evaluation, and threats to validity considered during study design and execution.
Results: Use of software patterns in maintenance is the most commonly investigated theme, explored in 16 studies. Object-oriented design patterns are evaluated in 14 studies while 4 studies evaluate architectural patterns. We identified 10 different constructs with 31 associated measures used to evaluate software patterns. Measures for 'efficiency' and 'usability' are commonly used to evaluate the problem solving process. While measures for 'completeness', 'correctness' and 'quality' are commonly used to evaluate the final artifact. Overall, 'time to complete a task' is the most frequently used measure, employed in 15 studies to measure 'efficiency'. For qualitative measures, studies do not report approaches for minimizing biases 27% of the time. Nine studies do not discuss any threats to validity.
Conclusion: Subtle differences in study design and execution can limit comparison of findings. Establishing baselines for participants' experience level, providing appropriate training, standardizing problem sets, and employing commonly used measures to evaluate performance can support replication and comparison of results across studies.

© 2015 Elsevier B.V. All rights reserved.

Contents

1. Introduction . . . 15
2. Background and related work . . . 16
   2.1. Software patterns . . . 16
   2.2. Mapping studies in software engineering . . . 16
   2.3. Replication studies . . . 16
   2.4. Secondary studies on software patterns . . . 17
3. Methodology . . . 17
   3.1. Inclusion and exclusion criteria . . . 17
      3.1.1. Exclusion criteria . . . 17

* Corresponding author.
E-mail addresses: mriaz@ncsu.edu (M. Riaz), breaux@cs.cmu.edu (T. Breaux), williams@csc.ncsu.edu (L. Williams).

http://dx.doi.org/10.1016/j.infsof.2015.04.002
0950-5849/© 2015 Elsevier B.V. All rights reserved.

3.1.2. Inclusion criteria . . . 17
3.2. Search and selection process . . . 17
3.3. Included papers . . . 19
3.4. Quality assessment of included studies . . . 19
3.5. Data extraction and analysis methodology . . . 19
4. Results . . . 20
4.1. RQ1: research questions and hypotheses . . . 20
4.2. RQ2: empirical design context factors . . . 20
4.2.1. Participant demographics . . . 21
4.2.2. Factors for grouping participants . . . 21
4.2.3. Preparatory material . . . 22
4.2.4. Task categorization . . . 23
4.2.5. Task details . . . 23
4.2.6. Patterns details . . . 24
4.3. RQ3: constructs and measures . . . 26
4.4. RQ4: threats to validity of included studies . . . 27
4.4.1. Internal validity of included studies . . . 29
4.4.2. Construct validity of included studies . . . 29
4.4.3. External validity of included studies . . . 29
4.4.4. Conclusion validity of included studies . . . 29
4.5. RQ5: replicated studies . . . 30
5. Discussion . . . 30
5.1. Identification of suitable participants . . . 30
5.2. Determination of expertise . . . 31
5.3. Standard problem sets . . . 31
5.4. Usage of preparatory material . . . 31
5.5. Study themes and evaluation constructs . . . 32
6. Threats to validity of our mapping study . . . 33
7. Conclusion . . . 33
Acknowledgments . . . 34
Appendix A. List of selected studies . . . 34
Appendix B. Quality appraisal of included studies . . . 35
Appendix C. Research questions and hypotheses of included studies . . . 36
References . . . 37

1. Introduction

While the availability of frameworks and tools enable developers to build more complex systems by reusing software components, software development remains a knowledge-intensive problem solving activity, largely relying on available skills and expertise [1]. Experts in any field are rare; capturing and sharing expert knowledge helps to raise the baseline performance of a novice (or non-expert) [2]. Software patterns are a popular approach for capturing and sharing expert knowledge for constructing successful solutions to recurring problems [3]. Despite the popular adoption of patterns in software engineering, how do we know that software patterns meet their goal of successfully capturing and sharing expert knowledge across the software development community to support problem solving? Applying software patterns during problem solving, much like most of software engineering, is a human-centered activity. The merit of a pattern can be assessed in terms of how successfully a practitioner employs a pattern to solve a particular commonly occurring problem. Empirical research evaluating the use of software patterns by software engineers (i.e., studies that involve human participants) can help in establishing whether a given pattern helps in problem solving.

The empirical evidence on how well various patterns help in problem solving is limited, often inconclusive, and merits further investigation. The context of empirical findings is also not well understood which limits applicability and generalizability of the findings. Furthermore, to lend validity to study findings and isolate confounding factors, empirical studies need to be replicated and synthesized. A clear description of the empirical protocol and a clear description of the context of the study is pre-requisite in replicating and synthesizing studies [4]. Systematic reviews, such as a mapping study, can help gather and categorize existing empirical research design practices, identify common research protocols, context factors and measures used to evaluate software patterns. Such information can be of use to researchers designing new studies or replicating existing studies.

To this end, the objective of this paper is to characterize the research design of empirical studies exploring software pattern application involving human participants.

We conducted a systematic mapping study [5] of existing literature on software pattern application in an empirical setting to aggregate existing research design practices. We characterize the research design in terms of the questions researchers have explored and the context of empirical research efforts including participants, patterns, training and tasks. We also classify the studies by what investigators' measured to evaluate pattern applications, and by threats to validity considered during study design and execution. We have established the following research questions to focus our analysis. We address each of these questions in the section referenced in parentheses.

RQ1. What research questions and hypothesis have been empirically investigated in reference to the application of software patterns? (Section 4.1).
RQ2. What is the context of empirical research efforts, such as participant demographics, preparatory material, patterns and task details, which investigate the application of software patterns? (Section 4.2).

RQ3. What constructs and associated measures have been used to evaluate the application of software patterns? (Section 4.3).
RQ4. What threats to validity related to the study design and execution have been considered by the researchers? (Section 4.4).
RQ5. How do replicated studies of software pattern application compare to original studies in term of study design? (Section 4.5).

Our analysis leads to considerations for mitigating some of the limitations of existing research to support replication and synthesis. We selected a set of 30 primary empirical studies reported in 27 papers for our analysis. The software patterns evaluated in the primary studies range from object-oriented design patterns to software architecture and security patterns. Object-oriented design patterns were inspired by Christopher Alexander's concept of patterns in building architectures [6] and are intended to be generic, reusable solutions that a software developer can "instantiate" into their system design to solve a problem [7]. These design patterns, popularly known as the Gang of Four (GoF) patterns, have received wide acceptance among practitioners and academicians and have led to the proliferation of patterns in other software engineering areas ranging from enterprise application architectural patterns [8] to software product line patterns [9] and security patterns [10]. In this paper, the term software pattern is inclusive of all pattern types used in the selected studies. Our research contributes the following:

- A systematic map of empirical literature that draws together research evaluating software pattern application in problem solving.
- Identification and categorization of data related to study design, including a classification of measures used to evaluate success when applying software patterns during problem solving.
- Compilation of reported threats to validity of studies evaluating software pattern application.
- Considerations to support replication and comparability of study findings.

The remainder of our paper is organized as follows: we discuss the background and related work in Section 2. In Section 3, we present our methodology for conducting the mapping study, including quality assessment of included studies. In Section 4, we present answers to our research questions. In Section 5, we discuss considerations based on our analysis. We discuss threats to validity of our mapping study in Section 6 and conclude our paper in Section 7.

2. Background and related work

In this section, we discuss software patterns and the role of mapping studies in software engineering as key background concepts. We also present background and terminology related to replication studies as we have included replication studies in our analysis. Other concepts and terminologies, such as types of threats to validity, are defined where discussed in the paper for clarity and readability. In terms of related work, we summarize existing secondary studies that cover various aspects of software patterns.

2.1. Software patterns

Software patterns are a means of providing successful solutions to common software problems [11]. Patterns are closely related to the concept of problem schemas. A problem schema is a mental structural representation that facilitates problem solving by grouping problems into categories that require similar solutions [12]. Similarly, software patterns represent a structural unit that captures the knowledge of an expert in solving common software problems that require similar solutions. Software patterns, like problem schemas, allow structuring of information such that the essential commonalities of the solution are abstracted, while leaving out specific details. Kohls et al. [13] have discussed this relation between design patterns and schema theory in detail, noting that patterns can be used to transfer knowledge and expertise by facilitating the creation of mental schemas in the recipient.

Jackson [14] argues that once the requirements, design and solution for a problem have been successfully formulated, it may be counterproductive to rework the problem from scratch. Software patterns essentially capture reusable solutions to common software problems that have proven to be successful over multiple instances. In this sense, software patterns are closely related to Jackson's problem frames [15]. A problem frame is a recognized class of problems specified primarily in terms of problem context, problem and solution domains, phenomena and requirements. The class of problems in a problem frame have a recognized solution. Problem frames can thus be considered problem patterns.

The software patterns community has attributed several benefits to the use of patterns during problem solving, such as: facilitation and ease of reuse; identification and capture of abstract concepts; aid in defining interfaces and interactions; means of shared documentation; construction of software with defined properties; and provision of common vocabulary [16]. Our analysis spans existing research on software patterns evaluating their success in providing these benefits.

2.2. Mapping studies in software engineering

Systematic mapping studies (or *scoping study*) provide a defined methodology to identify the scope and coverage of research results available in the literature on a software engineering topic of interest [5]. Mapping studies are differentiated from Systematic Literature Reviews (SLR) [17] based on the nature of research questions and the synthesis of collected evidence. Research questions in a mapping study are exploratory in nature and focus on providing insights into the current state of research and practice about a specific topic. Whereas, an SLR focuses on synthesizing results to address a well-defined research question [18]. The outcome of a mapping study is an identification of relevant studies and structuring of data from these studies based on the research questions. Additionally, mapping studies can be useful in identifying appropriate methodology for subsequent analysis of the data once the form and extent of dataset has been determined [16].

2.3. Replication studies

Replication studies are beneficial to evaluate the validity of prior study findings, either by reproducing results or by isolating factors that can influence results and that lead to variations. According to Lindsay et al. [19] a 'close' replication study attempts to recreate the known conditions of original study and is very similar to the original study. Close replications are often used to establish whether the original outcomes are repeatable at all i.e., initial outcomes were not unduly influenced by confounding factors. A 'differentiated' replication study has a known or deliberate variation in terms of a major effect of the original study conditions. Differentiated replications allow researchers to explore the impact of variations in treatments on the study outcomes [19]. If the replication study involves only a subset of tasks or studies a subset of factors, it will be considered a 'partial replication'. In addition to replication, Gomez et al. [20] argue that there are other ways of

verifying experimental findings, including reproduction and re-analysis of original studies. Where a replication study uses the same method as the original study, a reproduction study (also called a 'conceptual replication') uses a different method to reproduce the findings of the original study. A reproduction study can establish if the results of the original study are due to the study design or do the findings of the original study hold independently. A re-analysis study (also called an 'internal replication') uses existing data and re-analyzes it using similar or different analysis procedures to see if the results of the original study hold or to develop new insights based on the original data. We have included six replication studies in our analysis and use above definitions from literature to classify these replication studies.

2.4. Secondary studies on software patterns

A number of secondary studies have been conducted in recent years that assess the impact of design and architecture patterns on various software quality attributes.

Zhang and Budgen [21] conducted a mapping study to identify the usefulness and usability of GoF design pattern application in an empirical setting. With the exception of the role of design patterns in maintenance, Zhang and Budgen did not identify any firm support for the benefits attributed to design patterns based on their analysis of 11 selected papers. They suggest that future studies on empirical evaluation of design patterns should focus on the role of design patterns in maintenance.

Ampatzoglou et al. [22] have conducted a mapping study to investigate the effect of GoF design patterns usage on software quality attributes. They classify the studies into primary areas of research including formalization, detection, and application of design patterns. Based on the findings reported in primary studies, researchers found contradictory evidence related to the effect of design patterns on software quality. Researchers noted that design pattern detection and effect of design patterns on software quality are the most active areas of research related to GoF patterns. Ali and Elish [23] have also conducted a survey on the impact of GoF design patterns on software quality. They note that only a subset of Gof design patterns have been empirically evaluated and only four quality attributes have been explored related to GoF design patterns.

Galster and Avgeriou [24] have analyzed the impact of patterns related to the domain of service-oriented architecture (SOA) on various quality attributes. Researchers found a mismatch between patterns for SOA and important quality attributes for the domain of SOA.

Findings of the aforementioned secondary studies indicate an apparent lack of consensus regarding the impact of various patterns on software quality attributes. The findings also highlight a need for further empirical evaluation of software patterns while considering the context in which these patterns are applied to establish applicability and generalizability of the results. Our mapping study contributes towards mitigating some of the limitations identified in the above secondary studies by characterizing the research design of existing empirical studies on software pattern application and by providing considerations to support replication and synthesis. We aggregate and classify existing empirical literature to elicit common themes related to design, context and measurement of software pattern application. Thus, we do not consider studies that evaluate a software artifact in isolation [26], focus on structural or quality aspects of patterns in isolation [24], or use automated techniques for pattern detection. Furthermore, we explore the broader concept of software patterns, including GoF design patterns and SOA patterns considered in existing secondary studies.

3. Methodology

We have conducted a systematic mapping study to select and analyze existing literature [25] on the empirical evaluation of software pattern application by students and professionals. We present the details of our methodology below.

3.1. Inclusion and exclusion criteria

We are interested in empirical evaluation of software pattern application. In this section, we present the criteria used to identify publications related to empirical evaluation of software papers that are relevant to our research.

3.1.1. Exclusion criteria

Exclusion criteria are used to narrow down the initial set of search results by the process of elimination. Based on our research questions, we exclude publications that meet the following criteria:

(1) Publications that are unrelated to the software engineering domain.
(2) Books and non-peer reviewed publications on software patterns.
(3) A peer reviewed publication, in the field of software engineering, not related to empirical evaluation of software pattern application.

Our criteria will exclude publications that are not related to empirical evaluation of software patterns or do not involve application of software patterns.

3.1.2. Inclusion criteria

Inclusion criteria defines the attributes that are essential for a study to be selected in our analysis. Initially, we considered a stricter inclusion criteria indicating that studies should provide adequate details for various data elements related to empirical design. However no study was solely excluded on the basis of level of details provided. Publications that were not excluded per the exclusion criteria and that meet the following criteria are included in our analysis:

(a) Study describes an actual evaluation of software pattern application in empirical settings, not just a proposal, opinion or pilot study.
(b) Study provides some details related to the empirical protocol and context factors.
(c) Study provides some details related to the criteria for evaluating the outcome.
(d) Study provides some results and discussion based on the effect of using patterns.

Our inclusion criteria would exclude surveys where participants provide their opinion about software patterns [27] without using patterns in problem solving in an empirical setting.

3.2. Search and selection process

We conducted an exploratory search to identify appropriate search terms in accordance with the objective of our mapping study. For this purpose, we conducted search on Google Scholar using various combinations of terms 'software', 'pattern' and synonyms for 'study'. Given the broad meaning of term 'pattern' and the prevalent use of 'software' in diverse domains, we noticed a large number of results that were out of scope for our study. In

accordance with our exclusion criteria, we deemed two main types of results as out of scope: (1) literature covering domains other than software engineering e.g., patterns in biology, behavioral sciences, network traffic, etc.; and (2) literature on software patterns covering automated techniques for pattern application, mining, or evolution without involving human participants. Restricting the search to specific subject areas eliminated some of these results. The search results still included extraneous publications that were filtered out based on the inclusion/exclusion criteria.

During the exploratory search, we also noticed that many authors used more specific terms (e.g., 'architectural pattern', 'security pattern', 'requirement pattern') instead of the general term 'software pattern'. However such papers invariably mentioned 'design patterns' as a background concept. Many papers also discussed the introduction of 'patterns' concept from the field of 'architecture'. We also noticed that limiting the search to title, abstract, and keywords returned a very small number of results. Based on the results and insights from the exploratory search phase, we selected the following search term for our study:

- [[software AND pattern] OR [design AND pattern] OR [software AND architecture AND pattern]]
  AND
  [evaluation OR experiment OR empirical OR study]

With the selected term, we conducted search on four digital libraries for relevant papers up to September 2014. The search term could appear anywhere in the paper. The digital libraries are listed below:

- Google Scholar (Subjects: Engineering & Computer Science, Social Sciences).
- ACM Digital Library.
- IEEE Xplore.
- Science Direct (Type: Journals; Subjects: Computer Science, Engineering, Social Sciences).

We included 'Social Sciences' in the Science Direct subject because we were interested in the use of software patterns in education, among other areas, and this research can appear in Social Sciences publications (e.g., Computers & Education, Educational Psychology, etc.).

Table 1 provides a breakdown of our search results, removing duplicates and only considering papers that seemed potentially relevant given the scope of our study. We use two types of stopping criteria during our search: (1) we reviewed all results until we counted a fixed number of consecutive results that were out of scope (e.g., ACM and IEEE) and (2) we reviewed all results within a fixed-limit threshold (e.g., Google Scholar and Science Direct). We initially planned on using only the first stopping criteria. However we observed a large number of out of scope results early on in Google Scholar (the first digital library searched) and decided to continue looking at the results beyond the initial stopping criteria. We did not use any automated scripts or tools while searching the first three digital libraries. We walked through the search results in the order of decreasing relevance, as sorted by the digital library, marking potentially relevant papers in the process. In the case of Science Direct, we used EndNote[1] to import citations for first 1000 results returned based on relevance to our search string. These results were sorted in alphabetical order (on the name of each paper's first author) and examined in that order to identify potentially relevant papers.

Table 1
Search results by digital libraries.

Digital library	Search stopping criteria	Potentially relevant papers[a]
Google Scholar	First 1000 Results	409
ACM	40 consecutive results out of scope	83
IEEE Xplore	40 consecutive results out of scope	37
Science Direct	First 1000 Results	109
		Total: 638

[a] Excluding duplicates.

Table 2
Paper counts at each voting stage.

Voting stage	# of papers voted on by:				Selected for next stage
	Author 1	Author 2	Researcher 1	Researcher 2	
Paper titles	577	282	295	61	207
Abstracts	207	58	44	39	57

Our search queries produced an initial set of 638 potentially relevant papers, excluding duplicates. Four researchers separately voted on the paper titles during the first voting stage. The first author voted on all papers, as indicated in Table 2. The second author and two other researchers voted on subsets of 282, 295 and 61 paper titles, respectively. After applying the exclusion criteria, 207 papers were selected for voting on abstracts. The first author voted on all abstracts; the second author and two other researcher voted on a subset of 58, 44 and 39 abstracts respectively. We further eliminated 150 of these papers by evaluating the abstracts against the given inclusion and exclusion criteria, yielding a set of 57 papers to be considered in the next phase in which the whole paper is considered. At each stage, a paper with a positive vote from any of the voters was considered for the next stage.

Next, we performed a snowball search, looking at references of the 57 papers selected after voting on abstracts to identify any further potentially relevant papers. Three additional papers were found as a result, providing a total of 60 papers for the next step in the selection process. The first author applied inclusion criteria on the full papers in this set to yield the final set of 27 papers for our analysis.

To expand the scope of our search, we performed manual search on two journals, the Empirical Software Engineering Journal and Journal of Software: Evolution and Process. These journals are not indexed by the digital libraries used in our analysis but publish similar research. We found some papers on software patterns related to automatic pattern detection, pattern evolution in an artifact, or discussion of various structural or quality aspects of patterns without an empirical evaluation involving software pattern application. However, the search did not result in the inclusion of additional studies based on our criteria.

We used 11 papers on design patterns evaluation, identified in Zhang and Budgen's mapping study [21], as a quasi-gold standard [28] for our search process. Based on our search string, ideally the search results should included all of the 11 papers as we explore the broader concept of software patterns evaluation. We were able to identify 8 of the 11 papers based on our search string corresponding to a quasi-sensitivity of 72.72%. This is similar to the findings of Dieste et al. [29] for the digital libraries used in our mapping study and considered acceptable for Software Engineering reviews. Our overall search process was able to identify all 11 papers after the snowball search on papers selected after voting on abstracts.

[1] http://endnote.com/.

3.3. Included papers

Our analysis set includes a total of 27 papers covering software patterns from the domains including object-oriented design, software architecture, software security to collaborative learning and ubiquitous computing. Each paper is assigned a unique reference code for the purpose of our analysis. Twenty-one of the 27 papers document original studies whereas the remaining six report replication studies of one of the original studies. The 21 papers reporting original studies are assigned two-digit numerical codes 01 through 21 prefixed by the letter "S". The papers documenting replication studies are numbered differently, starting with the original study number followed by "-R" and a replication number (e.g. S04-R1 is replication one of study S04). We use these codes throughout the remainder of this paper to refer to individual papers. See Appendix A for the reference codes and bibliographical references of the included studies.

Paper S01 discusses the results of three distinct pattern-related empirical studies. We refer to these as S01-A, S01-B, and S01-C. Paper S03 describes two replications of a controlled experiment, which we refer to as S03-A and S03-B. In total, we examined 30 distinct primary studies (24 original studies; 6 replication studies) of software patterns reported in 27 papers. Five of these 30 studies are case studies and the remaining are experiments.

Studies S03 and S04 have been replicated one and five times, respectively. S03 was replicated by an independent set of researchers whereas a subset of researchers from the original study S04 was involved in all its five subsequent replications. Four of the replications of S04 have been conducted as part of a 'Joint Replication Project' [30] and are numbered S04-R2 to S04-R5. These replications were carried out using a replication web portal. Participants in all four of these replication studies were graduate and undergraduate students, whereas the original study was conducted on professionals. We discuss replication studies in Section 4.5 when we address RQ5.

Moreover, studies S03-A and S03-B can be considered close replications of the same experiment. Both experiments are conducted in the same timeframe, by the same researchers, and the conclusions are drawn based on the results of both studies. Studies S14 and S16 are based on the same experiment, but report findings on different factors that affect program maintenance in the presence of deployed design patterns. S16 can be considered a re-analysis study based on the same experiment as reported in S14.

With regard to the different types of replication studies, we are not making any assumptions about the variations we expect to see between original and replication studies. Any variations that exist between original and replication studies in terms of a study design aspect are reported within the context of the research questions. When summarizing our findings, we specify that these findings are based on X of 24 original studies and Y of 6 replication studies. If there are no variations between original and replication studies in terms of a study design aspect, we report the results in terms of original studies to avoid inflating the use of replicated design practices.

3.4. Quality assessment of included studies

For systematic literature reviews, Kitchenham and Charters [17] recommend quality assessment of included studies to assess the validity of the evidence as the evidence is synthesized to answer specific research questions. We assess the level of detail for the empirical protocol and various contextual factors provided in the primary studies, which is not typically required for mapping studies. For each paper, we consider the following five questions to assess this level of detail:

- Do the authors clearly state research questions or hypotheses for the study?
- Do the authors describe the sample population in terms of demographics and experience (including knowledge of patterns)?
- Do the authors explicitly describe the patterns they are using and rationale for selecting these patterns?
- Do the authors provide details about their findings and applicability of the findings?
- Do the authors discuss limitations/threats to validity of their study?

In answering each of the above questions, we use a scale from 0–3 (0 = Not at all; 1 = Somewhat; 2 = Mostly; 3 = Fully) as used by Kitchenham et al. [31] and compute the overall score by averaging the scores for individual questions. A higher quality score indicates that the researchers have provided a detailed report of the empirical study. Detailed reporting is important for supporting replication and analysis of a study. We provide the results of this quality assessment in Appendix B. Fig. 1 provides a breakdown of quality assessment scores over the years in which the primary studies are published. Studies often lacked in details related to the sample population and the threats to validity. However, almost 82% of the included papers have an average score over 2.5 indicating that the studies are of fairly good quality. The quality score for studies published in recent years is consistently high.

The quality scores assigned based on the above criteria reflect our perceived clarity and comprehensiveness of the presentation of information that we aimed to collect for our mapping study. These scores may not be reflective of the quality in terms of empirical design or of the publication venue.

3.5. Data extraction and analysis methodology

We have analyzed each study and extracted relevant data to answer our research questions. In Table 3, we list the data items extracted from each study.

We extract each of the above data item from each study and organize the data from different studies into common categories. In the spirit of Glaser, we perform grounded analysis to allow the structure, categories and themes to emerge from the data [32] while guiding our inquiry by our research questions. For instance, consider Table 6 listing factors for grouping participants. In the beginning, we do not have any category for the factors. From the first study, we identify that participants are grouped based on experience so we add a category 'experience-based'. In the second study, we find that grouping is done on the basis of experience as well as based on the differences in information and tasks. Thus we identify two additional categories 'pattern vs. non-pattern information' and 'pattern vs. non-pattern task'. We continue the analysis until we have gone through all the studies, placing studies either in already identified categories or creating new categories

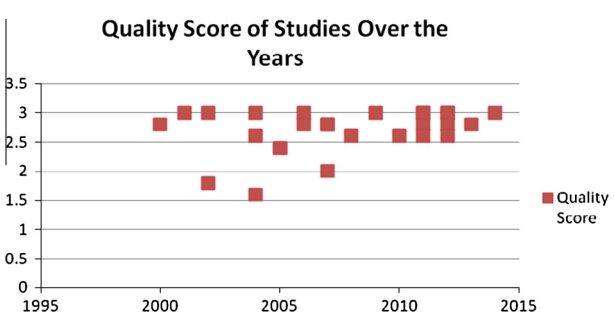


Fig. 1. Quality score of studies over the years.

Table 3
Extracted data items.

Category	Data Items
Research Questions	Research questions, hypothesis, assertions, constructs explored based on research questions
Participants	Role, experience level, method of determining expertise, incentive, sample size, grouping factors
Preparatory Material	Preparatory material, method of delivery of material
Tasks	Task type, lifecycle emphasis, number of tasks, magnitude of task, differences in tasks for participants, termination criteria for task
Patterns and Tools	Domain of patterns, patterns studied, rationale for selecting these patterns, programs, tools or frameworks used
Evaluation	Constructs used for evaluation, associated measures used for evaluation, approaches to minimize subjective assessment
Validity	Any reported threats to internal, external, construct and conclusion validity
Replications	Type of replication, motivation for replication, differences in findings between original study and replication

based on the data item. In some cases, we might have to combine multiple categories into a single category or divide into distinct categories to manage the level of details and clarity of information. For some data items, such as threats to validity, we have already established categories in the software engineering literature [33]. We use already established categories where applicable to categorize the data extracted during our analysis. Research questions RQ1–RQ5 are addressed in Section 4 by compiling and analyzing study design attributes of the included studies. Our findings are indicative of existing empirical design strategies among research practice.

4. Results

We present the results of our mapping study in this section. We report the details related to a replicated study separately only when differences exist between the original study and the replicated study in terms of a particular study design aspect.

4.1. RQ1: research questions and hypotheses

RQ1. What research questions and hypothesis have been empirically investigated in reference to the application of software patterns?

This research question directs us to understand the specific aspects of software pattern application that the researchers seek to explore. Research questions are interrogative statements that target a single phenomenon or concept and serve as a sign post to the investigator [34]. Empirical design and quantitative measurement begins with framing the research questions and deciding what to measure as dependent and independent variables. Hypotheses are falsifiable statements that link the independent and dependent variables and a statistically significant measure is used to attempt to disprove a hypothesis. A null hypothesis states that no statistically significant difference exists in a set of given observations. Whereas an alternate hypothesis is the negation of the null hypothesis. In Appendix C, we reproduce the research questions (prefix R_i) and hypotheses (prefix H_{0i} for Null Hypothesis; H_{ni} for Alternate Hypothesis), verbatim where reported. When a paper supplied none of these explicitly, we list the questions inferred from study description (*italicized*).

Table 4
Themes related to software patterns usage.

Theme	Studies focusing on the theme
Maintenance – modifying existing solutions	S02, S03-A, S03-B, S04, S08, S10, S11, S14, S16, S17
Construction – creating new solutions	S02, S06, S09, S12, S15, S20, S21
Training – learning problem solving in a new domain	S01-A, S01-B, S01-C, S02, S05, S06, S08, S13
Performance – providing better solutions	S04, S06, S07, S10, S12, S19, S21
Documentation – capturing or recovering rationales and decisions	S03-A, S03-B, S15, S17, S18, S20
Communication – supporting team communication	S06, S11, S21

Based on the research questions and hypothesis of the primary studies, we identify: (a) themes that the studies focus on in terms of software patterns usage and (b) constructs each study is interested in measuring. We identified six broad themes related to software pattern usage. If a study focuses on the role of patterns while modifying an existing solution, we categorize it as related to the theme of 'maintenance'. When new solutions are created, we categorize it as 'construction' theme. Studies that focus on training novices using patterns or introducing practitioners to new domain using patterns are categorized under 'training' theme. Studies where research questions explore how solutions involving patterns compare with alternate solutions are categorized as 'performance' theme. We list all the identified themes in Table 4 below. Note that each study can belong to multiple themes. Most commonly investigated theme, explored in 10 of 24 original studies, and all 6 replications, is the role of software patterns in maintenance-related activities.

Based on keywords in the research questions, we also identify the constructs each study is interested in measuring when using software patterns, as shown in Fig. 2. Constructs provide a granular view of what the researchers are interested in exploring as compared to the broad themes. In social sciences, constructs are social and psychological concepts that cannot be measured directly, but require measuring one or more variables indirectly [35]. Whereas height, weight and age are commonly accepted variables with standard units of measure, the concepts of productivity, usability and communicability are constructs that are measured indirectly. We organize the constructs as code-level, design-level and knowledge and communication. Some constructs may fall into multiple categories. For instance, 'design skills' is both a design-level and knowledge and communication construct. These constructs also help in identifying subtle differences between the studies. For instance, we can make a distinction between studies on architectural patterns exploring 'usability' vs. 'modularity' of the architecture. We may also compare constructs extracted from research questions with the constructs used to actually evaluate the outcome. Ideally these constructs should be similar so the goals, questions and metrics of the study are aligned [36].

4.2. RQ2: empirical design context factors

RQ2. What is the context of empirical research efforts, such as participant demographics, preparatory material, patterns and task details, which investigate the application of software patterns?

This research question directs us to identify 'who' are the participants (demographics, etc.), 'how' are they trained and assigned to the task (training, grouping, etc.), and 'what' task do they have to perform (problem, patterns, etc.). The context of empirical research efforts can help in establishing the applicability and generalizability of the findings and support future replications. We discuss each of the context factors below.

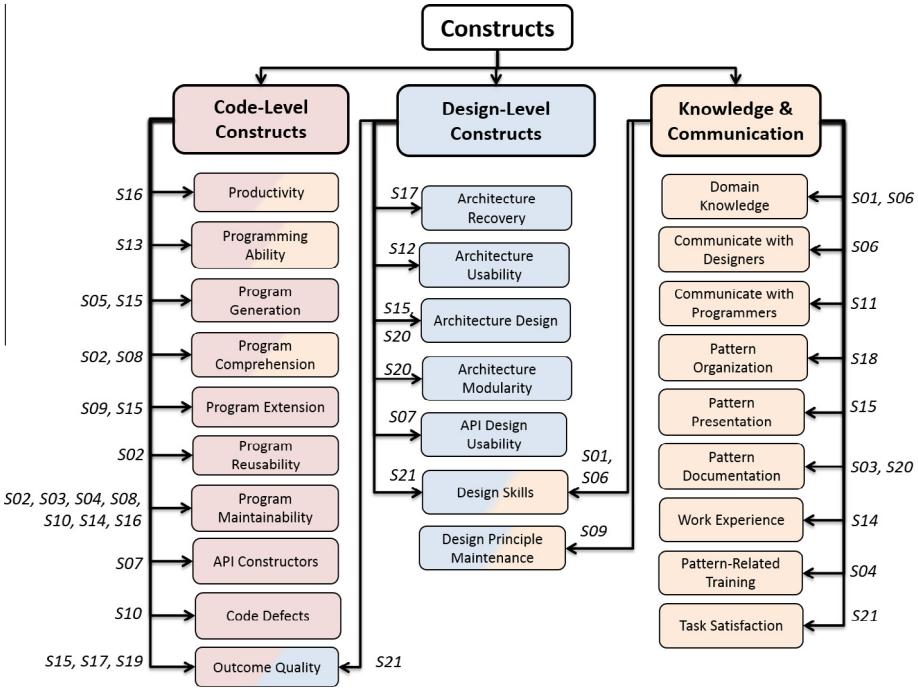


Fig. 2. Constructs hierarchy based on the research questions.

4.2.1. Participant demographics

Studies included in our analysis provide varying levels of details related to participants' role, experience level, and the type of incentive given to the participants. They also provide details related to the sample size of the participants. Table 5 provides a classification of the studies in terms of these demographic categories. We observed that 24 of the 30 empirical studies recruited undergraduate or graduate students enrolled in a computer science degree program. Four studies involved a mix of students and professionals while two studies recruited software professionals.

A majority of the included studies (16 of 24 original studies; all 6 replications) did not specify a discrete experience level for participants as researchers explored factors other than the role of experience level on study outcome. Six studies grouped participants into two discrete levels of expertise. Only two studies used more than two levels of expertise.

Study S04-R1 is the only study to offer monetary incentives to participants based on the participant's experience level, as rated by their employers. Credit for an assignment as part of a course was offered as incentive in 11 studies, considering that majority of the studies recruited students. Nearly half of the studies did not explicitly state the incentives given to participants.

We observed that 22 of 30 studies included in our analysis have 50 or fewer participants.

4.2.2. Factors for grouping participants

Studies in our analysis used different factors to divide participants into groups. The grouping criteria for each study is listed in Table 6.

Most commonly used criteria for grouping participants, used in 13 studies, is to provide one group of participants pattern-related information such as classes that participate in a particular design pattern. The other group is not provided any pattern-related information. Another commonly used factor for grouping participants, used in 9 studies, is to assign pattern vs. non-pattern tasks to different participants. For example, one group of participants is given a maintenance task on a program version containing design patterns, and the other group is given the same maintenance task on a different program version without design patterns. The performance of one group of participants is compared against the

Table 5
Details related to participants.

Participant Role	• **Undergraduates** (S01-A, S01-B, S03-A,S03-B, S03-R1, S04-R2, S04-R5, S05, S10, S13, S14, S16, S19) • **Graduate students** (S02, S03-A, S04-R2, S04-R3, S04-R4, S06, S08, S09, S11, S12, S14, S15, S16, S17, S18, S20) • **Mix of students and professionals** (S01-B, S06, S17, S21) • **Professionals only** (S04, S04-R1)
Experience Level	• **5 Levels:** 0–1 yrs, 1–2 yrs, 2–4 yrs, 4–6 yrs, 6 < yrs (S21) • **3 Levels**: Level 3 (20 + hrs training; limited practical experience), Level 2 (8–10 h training; no or some practical experience), Level 1 (2–4 h training; no or limited practical experience) (S01-A) • **2 Levels**: Expert vs. Novice (S01-B, S06, S11); Experienced vs. Inexperienced (S08, S14, S16) • **No discrete levels** (S01-C, S02, S03-A, S03-B, S04, S05, S07, S09, S10, S03-R1, S04-R1, S04-R2, S04-R3, S04-R4, S04-R5,S12, S13, S15, S17, S18, S19, S20)
Method of Determining Expertise	• **Self-assessed** (S04, S08, S17, S21) • **Test-based** (S03) • **Prior-performance based** (S13) • **Employer-rating based** (S04-R1)
Incentives	• **Volunteer** (S01-A, S04-R2, S04, S05, S07, S13, S17, S21) • **Assignment Credit** (S01-A, S01-C, S02, S04-R4, S04-R5, S09, S11, S16, S18, S19, S04-R5) • **Monetary** (S04-R1) • **Not specified** (S01-B, S03-A, S03-B, S03-R1, S04-R3, S08, S10, S12, S14, S15, S20)
Participant Sample Size	• **< 20** (S01-A, S01-B, S05, S07, S11, S12, S13, S15, S04-R2, S04-R3, S04-R4) • **20–50** (S01-C, S02, S03-B, S04, S06, S08, S17, S20, S03-R1, S04-R1, S04-R5) • **51–100** (S03-A, S09, S18, S21) • **>100** (S10, S14, S16, S19)

Table 6
Factors used to group participants across studies.

Factors for Grouping Participants	S01-A	S01-B	S01-C	S02	S03-A	S03-B	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21
Experience-based	X	X							X		X			X			X		X					
Individual vs. pair																								X
Patterns vs. non-pattern tasks		X		X			X			X	X		X				X		X					X
Patterns vs. non-pattern information		X			X	X			X	X	X				X	X	X		X	X		X		X
Less vs. more information			X												X			X			X		X	

other group of participants. To minimize presence of confounding factors, groups should be balanced in terms of other factors [37], such as background knowledge and experience, unless experience is also a factor in the study as is the case in seven studies.

Factors listed in Table 6 are between-subjects factors [38] in which different groups of participants receive different experimental treatments. Studies S02, S04 and S11 additionally compared performance of each participant pre and post-training related to software patterns. In these studies, pattern-related training is used as a within-subjects factor [38] in which same participant receives multiple levels of a treatment. Each participant is given multiple similar tasks to perform. However, some tasks are given before a course on design patterns and some afterwards. The performance of each participant in the later tasks is compared to their performance in the earlier tasks, prior to the course or training. To ensure that any observed improvement in performance is due to the training and not due to the participant gaining proficiency in performing the task, learning or practice effect should be countered [37]. One approach is to use counterbalancing by giving tasks in different orders to different participants.

Eight studies used multiple factors to group participants, with two studies (S04, S11) using both a between-subjects and within-subjects factor. Study S04 grouped participants based on patterns vs. non-patterns tasks. Additionally, participants performed the tasks before and after training related to patterns. Twenty studies used factors based on differences in information (last three factors in Table 6) to group participants. Two studies (S05, S09) did not divide participants into multiple groups. Study S05 explored the difficulties faced by novices when using design patterns. Study S09 reported a case study on how well participants, using state design pattern in their solution, adhered to the open-closed design principle.

Seven studies used participants' experience as a factor in study design i.e., researchers explored the difference in experience on task outcome in these studies. For these studies, we looked at the criteria used to determine different experience categories. As shown in Table 7, these studies used very different criteria to group participants as experts or novices based on the relative exposure of participants to the problem domain and task-related training.

The most pronounced difference between experts and novices was reported in study S01-B. In S11, difference between expert and novice was the least with experts having only spent one hour reading the requirements and design documents of the program under consideration.

Even where difference in experience is not used as a factor in the study design, researchers might consider the background and experience of participants to create balanced groups and to provide a reference for populations to which the findings may apply. In three studies (S04, S08, S17), participants were asked to self-assess their expertise levels, including familiarity with design patterns, by responding to a questionnaire. The self-assessments measure type and length of academic or work-related experience, thousand lines of code (KLOC) written, familiarity with problem domain, and the participant's self-perceived personal rating in comparison to other students or programmers. In these studies, the assessments were used to create balanced groups of participants. In study S03, a test assessing the participant's knowledge of patterns was conducted prior to the study. In study S13, investigators used the students' performance in a prior programming course to establish a performance baseline. In study S04-R1, employers rated the experience level of participants. Quantifying the experience level of participants in a standard manner across multiple studies is difficult in the absence of a gold standard [39]. In our analysis, the studies focus on different types of software patterns and problem domains, which further complicates the issue of identifying standard experience levels across studies.

Table 7
Criteria used to group participants where difference in experience is a factor in study design.

Reference code	Determination of Expertise
S01-A	• Level 1 (2–4 h training; no or limited practical experience) • Level 2 (8–10 h training; no or some practical experience) • Level 3 (20 + hrs training; limited practical experience)
S01-B	• Novices (undergraduate students, received first training in simulation environment) • Experts (simulation consultants, daily working with simulation environment)
S06	• Low experience (grad students with at most 3 person yrs of combined experience) • High experience pairs (professional/grad students with at least 6 person years of combined experience)
S08	• UML Knowledge (1: knows UML; 2: very good knowledge of UML) • Design Patterns Knowledge (1: knows design patterns; 2: very good knowledge of design patterns)
S11	• Novice (Other member of the pair) • Expert (1 h of studying requirements and design documents of program)
S14, S16	• Inexperienced (full-time UG, no formal work experience in industry) • Experienced (graduate students, at least 1 yr full-time work experience in industry)

4.2.3. Preparatory material

Investigators provide participants with preparatory material and activities to orient the participants to a task. In Table 8, we categorize the studies based on the preparatory material given to study participants. In many cases, software patterns were presented as part of the preparatory material to baseline participants with pre-requisite knowledge. Thirteen studies employ a short course to orient participants to the patterns. Four studies employ a long course. Studies employed more than one method for documentation and presenting patterns, and these methods are not mutually exclusive.

Three studies (S07, S12, S15) use an example from a real world system, such as an application programming interface (API) or

Table 8
Preparatory material and activities.

Method	Reference code	Example usage
Short course	S03-A, S03-B, S04, S05, S09, S10, S13, S14, S15, S16, S18, S19, S21	A tutorial session/short workshop to introduce the problem domain and patterns
Long course ($\geqslant$3 months)	S01-C, S02, S11, S20	An extended lab or semester long course focusing on the concept and use of software patterns
Reference material	S05, S06, S11, S12, S13, S15, S17	Paper-based/ online reference material containing information on patterns and pattern application provided to participants during the study
Pattern-related documentation	Design (S10); Code (S03, S14);	Design: comments about which patterns are used in design. Code: what are the classes that implement a pattern and what role they play in the pattern, or which member functions map to components of a pattern
Real world example	S07, S12, S15	Provide examples of pattern application in real world systems e.g., Factory pattern or Model-View-Controller (MVC) pattern implemented in Java APIs
Multiple examples (in varying context)	S07, S15	In addition to patterns, provide examples of concrete implementation of patterns in the problem domain in varying contexts
Pattern library	S01-A, S06	A computer or paper based pattern library or catalog that provides a list of all patterns related to domain
Video tutorial	S01-A	A short video about the working of various patterns
Expert guidance	S15	Availability of expert during the conduct of the study to provide onsite guidance
Annotated patterns	S18	Additional annotations specifying objectives, applicability, trade-offs and relationships among patterns when the original patterns either exclude or do not uniformly present such specifications
Practice session	S19	A session prior to the conduct of the study to provide a change to practice and ease participants into the actual experiment

architecture design, to orient the participants towards patterns. In study S15, an expert was present throughout the study execution to guide participants, answer any questions and resolve ambiguities. In study S19, a practice session was conducted prior to the start of actual study to help the participants understand how the experiment would be carried out.

Studies differ not only in the type of preparatory material given to the participants, but also in the timing of delivery and information content of the material. We observed four categories of methods used to deliver the patterns to study participants based on the timing and content of the material given:

- **Shared Introduction**. All participants receive the same preparatory material at the start of the study. This approach is used in 12 of 24 studies where researchers measure performance differences either due to existing differences in participants' expertise (e.g., expert vs. novice) or explore the performance of a homogeneous group of participants (e.g., novices) in using software patterns.
- **Shared Stimulus**. All participants receive the same material some time during the study execution, after having performed one or more tasks. This approach is used in 2 out of 24 studies to measure the impact of training on participants' performance, usually in a within-subjects design.
- **Varied Introduction**. Two groups of participants, wherein one group receives more detailed preparatory material than the other group at the start of the study. This approach is used in 6 of 24 studies to measure differences in pattern adoption and applicability based on the material provided.
- **Full Factorial**. Participants are sub-divided using all combinations of factors (E.g., preparatory material and participant expertise level) i.e., a full factorial experimental design. This approach is used in study S04, and its five replications, to measure the effect of each factor on the outcome.

4.2.4. Task categorization

In the studies, participants were asked to perform tasks related to software pattern application. We have identified three different categories of tasks based on lifecycle emphasis:

- **Pattern Selection Task** – participants select a relevant pattern to solve the problem and apply the pattern.
- **Pattern Instantiation Task** – participants are evaluated on their ability to apply a selected pattern.
- **Pattern Maintenance Task** – participants are asked to modify existing design or code with instantiated patterns, i.e., the investigators applied one or more patterns beforehand to the software design or code and presented that as part of the task. Understanding how patterns are instantiated in an existing system can help in identifying how to modify the system.

In terms of task type, some studies require participants to complete design tasks (D), others to implement code (I), and some require both (DI). Table 9 summarizes our analysis for lifecycle emphasis and task type. Assignment of studies to task categories is not mutually exclusive as a few studies examined both selection and instantiation of patterns.

Eleven studies focus on design tasks only. Twelve studies involve implementation tasks only. One study focuses on both design and implementation tasks (S02). Ten of the studies involving design tasks are carried out during pattern selection and instantiation. There is no specific lifecycle emphasis for implementation tasks.

4.2.5. Task details

In addition to lifecycle emphasis and task type, we report the categorization of included studies in terms of the details of tasks given to participants in Table 10. The number of tasks given to

Table 9
Task categorization (D: *Design*; I: *Implementation*; DI: *Design and Implementation*).

Pattern Task categorization	S01-A	S01-B	S01-C	S02	S03-A	S03-B	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21
Selection	D	D	I	DI				D	D							I		I			D		D	D
Instantiation	D	D	I	DI				D	D	I		I			D	I		I			D	D	D	D
Maintenance					I	I	I				D		I	I			I		I	D				

Table 10
Task details.

Task details	S01-A	S01-B	S01-C	S02	S03-A	S03-B	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21
Number of Tasks	3	1	1	2	2	2	9	3	2	5	2	1	3	2	1	6	3	2	3	1	5	2	1	1
Task Magnitude Measure																								
L: lines of code; **C**: # of classes; **P**: # of packages; **M**: # of methods; **X**: design complexity;				L	L + C	L + C	L + C			C	X	L	L + C + P	C + M	X		L + C + P		L + C + P					
Task Differences																								
N: no difference; **P**: pattern vs. non-pattern versions of same tasks; **D**: different problem sets; **S**: selected by participants;	N	P	S	S	N	N	P	N	N	P	P	N	D	N	N	N	P + D	D	P + D	N	N	N	N	N
Termination Criteria																								
T: time-limited; **O**: open-ended;	T	T	T	T	T	T	O	O	T	O	O	T	O	O	O	T	T	T	T	T	O	O	T	O

Table 11
Patterns details.

Category	Study-wise breakdown
Pattern domain	• Object-oriented (S02, S03-A, S03-B, S04, S05, S07, S08, S09, S10, S11, S13, S14, S16, S21) • Software architecture (S12, S15, S17, S20) • Software security (S18, S19) • Ubiquitous computing (S06) • Collaborative learning (S01-A, S01-C) • Simulation environment (S01-B)
Patterns; rationale for selecting patterns	• Ten ThinkLets [40]; Selection based on research, usage and expert opinion (S01-A) • Building blocks within Arena simulation environment [41]; Selection based on popularity and availability of simulation experts and vendors (S01-B) • Patterns for computer-mediated interaction [42]; Selected as part of lab curriculum (S01-C) • GoF Design Patterns (S02, S03-A, S03-B, S04, S05, S07, S08, S09, S10, S11, S13, S14, S16); Selected as part of course curriculum (S02, S09); Selection based on four most frequently occurring design patterns [43] (S14, S16); Selected to study properties of a particular pattern or set of patterns (S04, S07, S08, S09, S10); Used to study the effects of distributed cognition (S21) • Usability-Supporting Architecture Patterns (USAP); Selected based on the need to support a specific usability concern (S12) • J2EE Patterns, Architectural design patterns (S15, S17); Selected to support development of mobile applications (S15); Selected based on 'most well-known' architectural patterns (S17); Used to study patterns participants relationships (S20) • Security patterns; Selected to meet the needs of healthcare security and privacy scenarios (S18) • Pattern-based method for Secure Development (PbSD); Selected based on problem domain (S19)
Programs, tools and frameworks used	• A prototype tool to support selection of patterns (S01-A) • Arena simulation environment (S01-B) • Collaborative virtual learning environment CURE (S01-C) • And/Or-tree, Phonebook (S03) • Stock Ticker, Boolean Formulas Library, Communication Channel Library, Graphics Library (S04) • Interactive Quiz Environment (S05) • Eclipse IDE (S07) • Multiple-user Calendar Manager (S09, S10) • JHotDraw (S10, S14, S16) • Hotel Management System (S10) • ChiCo, Timme (S11) • J2ME, Agile Architecture Line Model (S15) • JBoss J2EE application server version 2.2.4 (S17) • Patient Monitoring System (S18) • Security Modeling Tool, Oracle VBD (S19)

the participants range between one and nine. Task magnitude measures reflect the measure used to specify the size of the programs on which tasks were performed. When stated, task magnitude measures provide additional details about the task and are not used to reflect differences in complexity of task between different groups. The number of tasks and task magnitude measures are not synonymous to task complexity in situations where participants receive multiple, small tasks or if the participants have to make a straightforward change in a large program.

In terms of task differences, for 14 studies, all participants received the same set of tasks to perform with variations in other factors such as the quantity and/or quality of available information, training or ordering of tasks. Studies S01-C and S02 allowed participants to select their own tasks as part of a semester-long project.

In terms of the termination criteria for task, 14 studies use a time-limited criteria (i.e., requiring participants to complete tasks within a predefined time), whereas 10 studies use open-ended criteria (i.e., allowing participants to take as much time as needed to complete the task).

4.2.6. Patterns details

The details related to software patterns include not only the domain of the patterns (e.g., object-oriented, software security) but also the exact patterns that were evaluated (e.g., GoF patterns). Additionally, to understand the context in which patterns are used

Table 12
Constructs and associated measures extracted from included studies.

Constructs	Associated measures	Example usage of measure in studies
Efficiency in problem solving	***Time to complete*** (S01-A, S01-B, S01-C, S03, S04, S06, S07, S11, S12, S13, S14, S16, S18, S19, S21)	To assess efficiency of experts vs. novices or pattern vs. non-pattern approaches. Often useful when comparing the times of correct solutions. Otherwise, need to account for time needed to fix errors
	Learning efficiency/Ease of understanding and performing the task (S01-A, S01-B, S01-C, S03, S06, S07, S08, S11, S13)	In an experiment with tasks of progressive difficulty levels, look at the spread of achievement among participants in a given time frame, using the knowledge of design patterns as the control variable
	Pattern selection ratio (S18)	The ratio of the number of patterns selected as part of the solution to the number of patterns considered for selection
Quality of solution	***Subject matter expert evaluation*** (S01-A, S01-B, S01-C, S05, S06, S17)	How do experts evaluate the overall quality of the deliverable using a scale (e.g., Likert-like or other interval scales)?
	As perceived by participants (S19)	Perceived quality of the pattern-based solution method by designers
	Adherence to design principles (S09)	How does a pattern (e.g., State pattern) help in complying with a design principle (e.g., open-closed principle)?
	Atomic design features (S21)	Decompose a design into atomic design features e.g., elementary classes and their relationships/associations
Correctness of solution	***Useful/working/accurate solution*** (S01-B, S01-C, S05, S06, S09, S10, S20)	In a maintenance task involving two functionally equivalent implementations with and without patterns, how useful is one approach in comparison to the other in assisting correct maintenance
	Number of errors (S03, S04, S10)	When performing a maintenance task, assess if use of patterns leads to fewer errors compared to maintenance task without employing design patterns; Assess difference in the number of errors made by novices using patterns vs. more experienced participants not using patterns
	Number of failing tests (S14)	In maintenance, identify functional problems in deliverable; in refactoring, ensure functional equivalence
Completeness of solution	***Requirements fulfillment*** (S01-B, S03, S06, S07, S09, S10, S12)	When performing maintenance tasks, how many of the required tasks were accomplished when using patterns vs. non-pattern approaches
	Recall and precision, as compared to a 'gold standard' solution (S19)	Looking at the granularity of access privileges in a database (row, column or table level) to see which approach provides more fine grained access
	Quantity of results (S17)	When recovering architectural decisions, how many decisions are recovered when explicitly considering or not considering patterns
Complexity of application	***Cognitive load***, qualitatively assessed (S01-A, S01-B, S01-C)	Based on the cognitive load theory, identify the type of load (intrinsic, extraneous, germane) and see if it is conducive to learning and developing knowledge schemas
	Eye focus, quantitative assessment of focus of attention during the task (S08)	Use an eye tracker to determine whether participants focused on the pattern-specific code while performing maintenance task or spent more time understanding non-pattern code
	Cyclomatic complexity, or number of linearly-independent paths (S15)	Compare cyclomatic complexity of deliverables to identify cases with complex components. Higher number correlates with higher risk of defects
	Ease of understanding the solution, qualitatively assessed (S20)	On a scale of 1–10, solution created using architectural patterns is more comprehensible as compared to alternative approach
Usability of patterns	***Pattern selection*** (S02, S05, S13, S19)	Assess the use of tools with support for pattern selection in helping participants
	Pattern application (S02, S05, S09, S10, S12, S13)	Assess the use of multiple examples showing how to instantiate given patterns to improve patterns applicability
	Task satisfaction measure, qualitative assessment (S21)	Participants' assessment of satisfaction while performing the task, on an established 7-point scale [44]
Communicability of pattern knowledge	***Oral communication*** (S06, S11, S15)	How do pairs communicate with and without design patterns knowledge?
	Written Documentation (S15, S20)	How does patterns knowledge aid in capturing design rationales?
Creativity of solution	***Subject matter expert evaluation*** (S06)	Human computer interaction experts assess the creative aspects of a deliverable
Modularity of solution	***Coupling*** (S02)	Computed based on the deliverables in a programming task
	Cohesion (S02)	
	Weighted Methods per Class (S02)	
	Architecture decomposition (S20)	Assessment of how effectively the software architecture can be decompose into components/sub-components to assign responsibilities
Size of solution	***Lines of Code*** (S02)	Compare pattern vs. non-pattern to see which solution has a larger size. Pattern-based solutions are mostly found to be larger in size as they introduce class hierarchies and abstract classes or interfaces
	Number of Classes (S02)	
	Number of Operations (S02)	
	Number of Attributes (S02)	

in a study, it is important to document the rationale for selecting particular patterns and the tools or frameworks within which the patterns were applied. We list the details related to the software patterns evaluated in the included studies in Table 11. Not all studies specified the exact patterns that were studied, which can limit the applicability of the study findings and preclude future replications.

Fourteen of the 24 included studies (and all six replication studies) evaluate the use of object-oriented design patterns, specifically the GoF patterns. Four studies evaluate software architectural patterns, with two of these studies looking at J2EE patterns. No other patterns are explored in multiple studies. No particular tool or framework is used across multiple studies except for JHotDraw[2]

[2] http://www.jhotdraw.org/.

application employed in three studies, all conducted by the same group of researchers.

4.3. RQ3: constructs and measures

RQ3. What constructs and associated measures have been used to evaluate the application of software patterns?

This research question directs us to identify and categorize the commonly used constructs and measures. Categorizing commonly used measures makes it more likely that same measures would be used for the same construct in similar situations, supporting comparison and synthesis of findings. We have categorized the included studies in terms of 10 different constructs with 31 associated measures based on the evaluation criteria used to assess the outcome, as given in Table 12. The examples given in the last column are pulled from different studies and illustrate the use of a particular measure. Studies in our analysis often used multiple measures, 3–4 different measures on average. In Fig. 3, we summarize how frequently each of the 10 constructs is used in the 21 papers reporting original studies.

Efficiency is the most commonly used construct, followed by correctness and completeness, measured in 9 studies each. Usability of patterns is explored in eight studies.

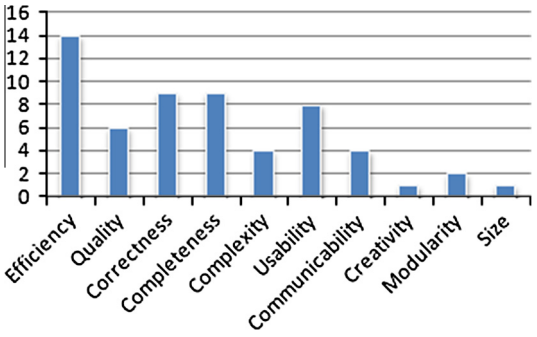


Fig. 3. Frequencies of constructs used in the primary studies.

In our analysis, the measures used for different constructs include both quantitative measures (e.g., time to complete a task) and qualitative measures (e.g., subject matter expert evaluation). Also, the measures are used to evaluate both the final artifact produced as an outcome of the study and the process involved in producing the artifacts. In Table 13, we provide a categorization of the measures in terms of whether (a) measure is qualitative or quantitative and (b) measure evaluates the artifact or the process to produce the artifact. The measure 'Eye focus' can be considered as a hybrid measure, both quantitative and qualitative, because the tool results in a point of eye focus, but the investigator maps the point to a feature, which is a subjective interpretation. We also report how frequently each measure is used in the included studies in Table 13. Measures for 'efficiency' and 'usability' are most commonly used to evaluate the problem solving process. While measures for 'completeness', 'correctness' and 'quality' are most commonly used to evaluate the final artifact. The most frequently used quantitative measure, used in 15 studies, is the 'time to complete a task' whereas 'learning efficiency' is the most prevalent qualitative measure, used in 9 studies, when evaluating software patterns for efficiency in problem solving.

Intuitively we would expect that studies involving human participants would have most of the measures related to the problem solving process itself. However in our analysis, almost two-thirds (21 of 31) of the measures are related to the final artifact as the artifact is a reflection of how well the participants are able to use the patterns in problem solving. Although we have fewer measures (10 of 31) to evaluate the problem solving process itself, these measures are used repeatedly across studies. The two most frequently used measures, 'time to complete' a task and 'learning efficiency' both measure the process, indicating that researchers of the studies are often interested in evaluating how software patterns support the problem solving process. We found an equal number measures that are quantitative and qualitative.

Three qualitative measures are self-reported by the participants and are inherently subject to personal opinion of the participant. However, for the remaining 13 qualitative measures that are computed by the researchers, we assessed if the researchers of the studies used approaches to reduce subjectivity and potential biases, such as measuring inter-rater agreement [45] or using double-blinding. In Table 14, we report the studies that employed

Table 13
Categorization of measures with frequency of use (bold-highlight indicate self-reported measures).

	Quantitative Measure		Qualitative Measure	
Artifact Measure	Completeness	• Requirements fulfillment (7) • Precision/Recall (1) • Quantity of results (1)	Quality	• Subject matter expert evaluation (6) • **Perceived by participants** (1) • Adherence to design principles (1) • Atomic design features (1)
	Correctness	• Number of errors (3) • Number of failing tests (1)	Correctness	• Useful/Working solution (7)
	Complexity	• Cyclomatic complexity (1)	Complexity	• Ease of understanding the solution (1)
	Modularity	• Coupling (1) • Cohesion (1) • Weighted methods per class (1)	Modularity	• Architecture decomposition (1)
	Size	• Lines of Code (1) • Number of classes (1) • Number of operations (1) • Number of attributes (1)	Creativity	• Subject matter expert evaluation (1)
Process Measure	Efficiency	• Time to complete (15) • Pattern selection ratio (1)	Efficiency	• **Learning efficiency/Ease of understanding** (9)
	Complexity	• Eye focus (1)	Complexity	• Cognitive load (3) • Eye focus (1)
			Communicability	• Oral communication (3) • Written Documentation (2)
			Usability	• Pattern selection (4) • Pattern application (6) • **Task satisfaction measure** (1)

Table 14
Approaches to minimize subjectivity when using qualitative measures.

Criteria	Measure	Approaches to Reduce Subjectivity
Quality	• Subject matter expert evaluation	• ***S01-A:*** evaluated by two researchers based on a quality checklist, report inter-rater reliability based on Pearson correlation • ***S01-B:*** evaluated by multiple experts and problem owners • ***S01-C:*** no details provided • ***S05:*** evaluated by researchers, no information given • ***S06:*** use of multiple evaluators (1 HCI TA, 2 ubicomp researchers); evaluators blind to treatments (patterns vs. non-patterns) • ***S17:*** use of 2 professionals as evaluators; 5 point scale; based on a checklist of parameters; double-blinding (decisions in outcome not grouped, evaluators not aware of which decisions go together, which decision belong to which participant or existence of treatments); test hypothesis for individual evaluators and average of evaluators
	• Adherence to design principles	• ***S09:*** measure adherence to open-closed principle when applying the state design pattern against a pre-implemented model solution
	• Atomic design features	• ***S21:*** create an expected standard for design solution to evaluate the responses; two independent evaluators and check for inter-rater consistency; raters blind to treatment
Correctness	• Useful/Working Solution	• ***S01-B:*** evaluated by multiple experts and problem owners • ***S01-C:*** participants voted to decided best solutions • ***S05:*** use of solution oracle to assess correct selection and expected design, no details on how the solution oracle was created • ***S06:*** use of multiple evaluators (1 HCI TA, 2 ubicomp researchers); evaluators blind to treatments (patterns vs. non-patterns) • ***S09:*** no details provided • ***S10:*** no details provided
Creativity	• Subject matter expert evaluation	• ***S06:*** use of multiple evaluators (3 HCI graduate students); evaluators blind to treatments (patterns vs. non-patterns); evaluators shown outcomes to be evaluated in different order
Complexity	• Cognitive load	• ***S01:*** no details provided
	• Eye focus	• ***S08:*** Use of eye tracker with maximum available resolution and fastest data rate for head-mounted eye trackers; cross-verified eye-tracker data with video recordings of eye movements; discarded data for one participant due to decalibration of eye-tracker
	• Ease of understanding the solution	• ***S20:*** evaluated by three experts with shared understanding of the measure; use of *t*-test to check for similarity in mean assessment and variance of each evaluator
Communicability	• Oral communication	• ***S06:*** video tape - no info given on subjectivity • ***S11:*** transcribe the communication between participants in a pair and code the transcription in terms of pre-defined codes (coding done by one researcher only) • ***S15:*** no details provided
	• Written documentation	• ***S15:*** participants captured rationale behind the design decisions so that it is easy for others to understand (no inter-rater agreement given)
Usability	• Pattern selection	• ***S02:*** reported by participants • ***S05:*** evaluated by researchers, no details provided • ***S13:*** compare the maximum difficulty level of task completed by participants when tasks were of progressing difficulty • ***S19:*** compared against a 'gold standard'
	• Pattern application	• ***S02:*** reported by participants • ***S05:*** evaluated by researchers, no details provided • ***S09:*** no details provided • ***S10:*** no details provided • ***S12:*** create an oracle of all possible responsibilities that can be considered by taking a union of responsibilities considered in the participants' responses • ***S13:*** compare the maximum difficulty level of task completed by participants when tasks were of progressing difficulty
Modularity	• Architecture decomposition	• ***S20:*** evaluated by three experts with shared understanding of the measure; use of *t*-test to check for similarity in mean assessment and variance of each evaluator

approaches to reduce subjective assessment of qualitative measures.

Based on Table 14, two common approaches to reduce subjective assessment are the use of multiple evaluators to evaluate each outcome, and the use of a solution oracle or checklists to compare the outcomes against. However, inter-rater agreement was reported in only two of the ten cases that involved multiple raters. The mechanism to compose the solution oracle or checklists was also often unclear, documented in only one of the five cases. Studies did not discuss the issue of minimizing subjective assessment 9 out of 33 times.

4.4. RQ4: threats to validity of included studies

RQ4. What threats to validity related to the study design and execution have been considered by the researchers?

This research question directs us to identify and categorize the threats to validity that have been considered and mitigated at various steps by the researchers. Threats to validity of empirical evaluation can be mitigated by case study or experiment design. In situations where designs cannot mitigate threats, those threats must be reported as potential limitations. Twenty-one (21) studies discuss one or more of the four general types of threats to validity that will be discussed in the following subsections and categorized in Table 15. We have used the established categories for types of threats to validity as listed by Wohlin et al. [33] whenever applicable.

While there can be additional, unidentified threats to validity not considered or addressed by the authors, we are primarily synthesizing and reporting the threats that are explicitly considered. Identifying a relatively complete list of threats to validity of included studies is beyond the scope of this mapping study.

Table 15
Threats to validity of included studies.

Validity type	Threat type	Example of considerations for a given threat type
Internal validity	Selection (S03, S04, S09, S10, S04-R1, S11, S17, S18, S19, S20, S21)	Are there variations in expertise of the selected participants? Are the participants or participant groups appropriately balanced by expertise, if assumed to test the hypothesis?
	Mortality, or participant attrition (S03, S04, S14, S04-R3)	Do participants exiting the experiment before completion cause the groups to become imbalanced, which leads to missing data points?
	Testing and training (S19)	Have the participants received adequate training using quality preparatory material that can lead to observable differences in using pattern vs. non-pattern approaches?
	Diffusion or imitation of treatments (S08, S09, S10, S14, S16, S17, S18, S04-R2, S04-R4)	How to minimize undesired information flow among participants? If tasks are given in a different order, or if participants are divided in control and treatment groups, they should not interact with each other and exchange information during the experiment
	Resentful demoralization (S03, S04-R1, S16)	Are slower participants feeling time pressure, whether due to an approaching deadline or because other participants are leaving the experiment early?
	Instrumentation (S04, S04-R1, S04-R2, S04-R3, S04-R5, S08, S16)	Are measurements of time spent on each task precise and accurate? If a special technology is used, such as an eye tracker to measure focus, is the technology correctly calibrated and are the measurements reliable?
	Maturation, or learning and fatigue effects (S07, S08, S11, S14, S04-R4)	If the tasks are presented in a specific order, does changing the order lead to different findings?
	Interactions with selection (S04-R1, S21)	If a specific language is used, such as C++, Java or Python, does this affect the results significantly if all other factors are controlled among the selected participants?
	Presence of un-identified patterns (S14, S16, S17)	Are intended patterns the only patterns present in the task, or are there other, un-identified patterns that may impact measurements?
	Compensatory equalization of treatments (S14, S16, S18, S04-R5)	If the experimental exercise covers multiple, functionally equivalent program versions, such as one version with patterns and another without patterns, how is functional equivalence ensured?
Construct validity	Mono-operation bias (S08, S17)	Are the programs selected from a variety of problem domains to minimize any bias introduced due to familiarity of certain participants with specific domains, skewing the results in favor of some participants?
	Mono-method bias (S08, S17)	Are multiple types of measures or observations used to avoid measurement bias and misleading results?
	Hypothesis guessing (S04-R5, S08, S19)	Can the participants anticipate the expected results, which may cause the findings to better or worse reflect the expected findings?
	Evaluation apprehension (S04-R1, S04-R4, S08, S16)	Are participants exposed to experimental apprehensions or stressors, such as lack of motivation, fear of poor performance, or another liability, that hinders their potential to perform normally?
	Observable measures used (S04-R4, S04-R5, S17, S21)	Are multiple comparable measures used to evaluate a particular criteria, such as code quality, such that results across different measures can be crosschecked?
External validity	Representativeness of training and preparatory material (S02, S04-R5, S18)	Is the instructor's teaching style sufficiently generic to communicate knowledge and course content in a manner similar to other instructors?
	Representativeness of sample population (S02, S03, S04, S04-R2, S08, S09, S11, S16, S17, S18, S19, S20)	How closely do study participants represent the target population to whom inferences will be made?
	Uncontrolled task selection (S02)	When study participants choose their own tasks, such as when students select a semester-long project, are all these self-selected tasks comparable and findings generalizable in terms of problem domain, complexity and outputs?
	Experimental constraints that limit realism (S03, S04, S04-R1, S11, S17, S19, S20, S21)	How closely does the experiment setup represent real-world situations? For example, are participants asked to work alone in the study, when the real world setting permits or requires teamwork? Are participants required to manually perform a task, whereas a real world setting employs automated techniques or tools?
	Generalizability to larger and more complex tasks and programs (S03, S04, S04-R1, S04-R2, S07, S10, S11, S14, S16, S18, S19)	Do the findings from a focused, controlled task generalize to larger, more complex tasks? Do the findings from small, relatively simple programs generalize to larger, more complex programs?
	Program and task representativeness (S03, S04, S07, S08, S09, S10, S14, S18)	Are the programs and tasks given as part of the experiment representative of real-world programs? For example, if maintenance tasks are given, are these tasks representative of the range of possible maintenance task types (e.g., adaptive, perfective, corrective)?
	Pattern representativeness (S04, S10, S14)	Are the patterns used in the experimental setup representative of other patterns?
	Tool and language representativeness (S04-R1, S14, S16)	If a specific programming language or platform is used, can the findings be generalized to other languages and platforms?
Conclusion validity	Random irrelevancies in experimental setting (S04-R5, S08)	Are there any problems in experimental settings (e.g., surrounding noise) that can introduce error and interfere with the findings?
	Insufficient number of participants or data points (S03, S08, S14)	Have a sufficient number of participants and data points been collected to draw reliable conclusions? Otherwise the power of the experiment will be lowered.
	Low statistical power (S09, S14)	What is the probability of not accepting the alternative hypothesis when it is true?
	Reliability of measures (S03, S04-R5, S16, S17, S19, S20)	Are the quality assessments realistic? Can objectivity be increased, such as by using multiple independent raters, acceptance tests, comparing only correct solutions, or accounting for time needed to fix errors?
	Violated assumptions of statistical tests (S17, S20)	Is appropriate type of analysis carried out based on the type of scale so that correct statistical inferences could be drawn?

Table 15 should not be considered as a complete list of all the potential threats to validity of the included studies.

4.4.1. Internal validity of included studies

Internal validity is the degree to which an investigator can correctly draw causal inferences from empirical data. Issues related to experimental procedures, treatments, and participant experiences may threaten internal validity [34].

The two most commonly reported threats to internal validity, discussed in 11 and 9 studies respectively, are: selection bias due to variation in expertise among participants and treatment diffusion. Selection bias due to variation in expertise is mitigated by employing counter-balancing or random block assignment. Counter-balancing reduces the effect that a specific factor (e.g., expertise, tasks) may have on a group's measures. For instance, study S03 employs counter-balancing to distribute demographic factors across the treatment and control groups. Study S04 uses random blocked assignment to explicitly balance the groups in terms of background and expertise. Additionally, in S04, investigators computed that only 7.5% of the variation in results could be attributed to differences in participants' expertise level, further demonstrating that this particular threat was minimal and potentially mitigated by the counter measure (random block assignment). In both studies, expertise level of each participant was assessed prior to the study and factored in while assigning participants to each group.

Other commonly considered threats to internal validity include whether treatments have been equalized across groups (4 studies), learning or fatigue effects when performing multiple tasks in a sequence (5 studies), and potential errors in instrumentation such as accuracy in measurement of time (7 studies).

4.4.2. Construct validity of included studies

Construct validity is the degree to which the experiment is measuring what it is purported to measure [34]. The use of ambiguous definitions that lead to multiple, even conflicting, interpretations of the experimental construct is a threat to construct validity.

Evaluation apprehension, a human tendency to try to look better or fear of being evaluated, is the most commonly identified threat to construct validity as reported in four studies. Participants may experience apprehensions during the experiment, such as fear of poor performance. These apprehensions can hinder participants' potential to perform to the best of their abilities whether they are based on actual or perceived liabilities associated with participation in the experiment. Measuring performance of participants in the presence of such unintended apprehensions can lead to an inaccurate assessment of participants' abilities, which can threaten construct validity. Blinding can minimize biases and apprehensions on part of the experimenter and participants [46], leading to increased validity of the findings. In study S08, double blinding was used in which participants were not informed about the goal of the study and a level of indirection was used so that results could not be traced back to any specific participant.

An important consideration is to choose valid measures for each construct, as considered in four studies. Quantitative constructs, such as code size, are easier to measure, whereas qualitative constructs, such as quality and efficiency, are prone to variable interpretations and require the investigator to provide an explicit account of the assumptions that link the measures to the construct. In qualitative case study research, strategies to mitigate threats to construct validity include using multiple sources of evidence, establishing chains of evidence and using key informants to review the study findings [37].

Hypothesis guessing is another commonly considered threat to construct validity (3 studies). If participants can anticipate the expected results, their performance can be biased in favor or against the hypothesis, threatening construct validity.

4.4.3. External validity of included studies

External validity is the degree to which the results are generalizable to other people, situations or past, present and future events [34].

Two commonly identified threats to external validity, reported in 12 and 11 studies respectively, are representativeness of sample population and task generalizability. Generalizability of results is limited in the absence of explicit theories of software construction and maintenance. Generalizability is further constrained by the considerable variations among patterns.

In 4 of the 30 studies, the research questions explicitly focus on teaching design skills to students, which is the participant population of interest. For the remaining studies, the research questions of these studies (as listed in Section 4.1) and the nature of tasks given to the participants (e.g., development tasks, maintenance tasks, tasks related to software architecture) indicate that the sample would ideally be drawn from a population of professional software engineers of varying experience and expertise to improve the representativeness of sample population. Although students may be used instead of professionals under certain circumstances [47], further research by empirical software engineering community can guide how well results from students can be generalized to professional software engineers.

Representative of the program and tasks (8 studies) given as part of the study and experimental constraints that limit realism in general (8 studies) threaten external validity of the findings. Representativeness of training materials used (3 studies), patterns studied (3 studies), and tools and languages (3 studies) are also considered as threat to external validity. In many cases, these threats cannot be directly addressed as experimental setting is often a scaled-down version of real-world problems in a controlled environment. However, creating a repository of standard problem sets can help minimize potential biases.

4.4.4. Conclusion validity of included studies

Conclusion validity is the degree to which investigators draw statistically accurate conclusions from the data, for instance by having sufficient statistical power and by satisfying all statistical assumptions [34]. The power of an experiment represents the probability of rejecting a null hypothesis when it is false during statistical analysis of the results [48].

Reliability of measures (such as subjective assessment of outcome) was the most commonly identified threat to conclusion validity, as reported in six of the studies. Another issue that threatens conclusion validity is the need to have a sufficient number of participants and data points from which to draw reliable and statistically significant conclusions. Moreover, appropriate type of analysis must be carried out based on the type of measurement scale used so that correct statistical inferences can be drawn. Using incorrect type of analysis can threaten conclusion validity, as discussed in S17.

Low statistical power (2 studies) or violation of underlying statistical assumptions (2 studies) during analysis can invalidate the conclusions drawn as part of the study. Random irrelevancies (2 studies), such as noise or stressful environment, can also interfere with performance and limit validity of the findings.

4.5. RQ5: replicated studies

RQ5. How do replicated studies of software pattern application compare to original studies in term of study design?

This research question directs us to classify the replication studies and identify factors that may be conducive to replication. Two studies included in our analysis, S03 and S04, have been replicated one and five times respectively. We are interested in identifying how similar the replications are to the original studies in terms of study design. We classify the replication studies according the definitions established in the literature (see Section 2.3) as shown in Table 16. All the replications are differentiated replications. Three replications only involve a subset of tasks as compared to the original study. In one case, the replication study examines additional factors than the ones explored in the original study. In all other cases, replication studies explore the same factors.

In general, replication studies consider the same constructs as the original study however they may use an alternate or additional measure. For instance, S04-R4 defined its own scale for correctness for one of the experiments in the study but used the same scale for the replicated experiment. Of the six replication studies, only S04-R4 considered an additional factor in that participants were familiarized with the graphical design of the software before carrying out the tasks. Findings of the replication of S03 are consistent with the original study that pattern documentation in code is useful in increasing the efficiency during maintenance activities. However, findings of replications of S04 differ from the original study and are inconclusive in establishing the usefulness of various design patterns.

Both the studies that are replicated have a quality score of 3 (Appendix B), indicating that the researchers have provided a detailed report of the original study. Of the 24 original studies, only 5 others have a score of 3 based on our quality assessment. We list factors that might be conducive for replication, as identified from the original and replication studies, in Table 17. Availability of either a replication package, including study materials, or involvement of one or more of the original researchers may also support the replication effort.

Four replications of S04 (S04-R2 to S04-R5) are carried out as part of a joint replication effort. S04-R2 involves one of the original researchers who loosely collaborate with the rest of the researchers (S04-R3 to S04-R5). The purpose of a joint replication is to collect sufficient data points for joint evaluation and analysis. All the four replications are carried out using a replication web portal to conduct the experiment and collect results however a joint evaluation of the results has not been reported yet.

Table 17
Factors conducive for replication.

Factors conducive for replication	S03-R1	S04-R1	S04-R2	S04-R3	S04-R4	S04-R5
Availability of a replication package	X					
Detailed reporting of original study	X	X	X	X	X	X
One or more of the original researchers involved in replication		X	X			
Joint replication effort			X	X	X	X

5. Discussion

We want to promote empirical evaluation of software patterns. Ability to replicate the studies and synthesize different outcomes is an important aspect of empiricism. For this, we need to understand and account for the variations in the context variables. Based on our analysis, we have identified considerations that can support replication and comparison of findings across studies as discussed in the following subsections.

5.1. Identification of suitable participants

In terms of participants' demographics, students often provide a sample of convenience for empirical software engineering research, as observed in our analysis as well. However, students may be a suitable demographics based on the study goals. For instance, eight studies explicitly relate to the theme of training using patterns as listed in Table 4. Novices, such as students or software professionals new to the problem domain, form a suitable demographic in these studies. Graduate students often have a fair amount of industry experience and can be representative of junior software practitioners. For instance in S09, participants had an

Table 16
Classification of replication studies.

Ref. code	Classification	Details
S03-R1	Differentiated Replication	• *Difference in empirical setting:* S03-R1 is a web-based activity whereas the original experiments (S03) were paper-based. Additional variations are related to participants' demographics and structuring of pattern-specific documentation used in the replication study. S03-R1 is also an independent replication as it is performed by a different set of researchers than those involved in the original study • *Difference in participants' demographics:* Participants in replicated experiment are undergraduates whereas original experiment involved both undergraduate and graduate participants
S04-R1	Differentiated Replication	• *Difference in empirical setting:* The setting of the replication study was closer to a real programming environment (tasks were carried out on a computer instead of paper; participants were selected from multiple consultancy companies; participants were provided monetary compensation for participation) as compared to the original study S04 • *Difference in participants' demographics:* Participants in both studies are professionals. However, expertise is self-reported in the original study whereas it is based on employer's rating in the replication
S04-R2, S04-R3, S04-R5	Differentiated (Partial) Replications	• *Subset of tasks and factors:* The replications involve only a subset of tasks and consider a subset of factors (dropping out the effect of training) when compared to the original study S04 • *Difference in empirical setting:* Tasks were carried out on a computer instead of paper. Original study uses C++ programming language whereas replicated studies use C++, Java or C# • *Difference in participants' demographics:* Participants in replicated experiments are graduate and undergraduate students whereas participants in the original study are professionals
S04-R4	Differentiated Replication	• *Additional factors studied:* S04-R4 introduces an additional factor (availability of graphical design of the programs) which may have impacted the outcome • *Difference in empirical setting:* Tasks were carried out on a computer instead of paper. Original study uses C++ programming language whereas replicated studies use C++, Java or C# • *Difference in participants' demographics:* Participants in replicated experiments are graduate students whereas participants in the original study are professionals

average industry experience of five years. In other cases, students may be substituted for professionals depending on the nature of the task. Host et al. [47] found only minor differences between students and professionals in carrying out various software engineering tasks requiring "general understanding of dependencies and relationships", such as project impact assessment. Students may be used in similar tasks in place of professionals. Carver et al. [49] have provided a checklist to consider when selecting students as study participants. They also highlight requirements related to ethics, validity, and generalizability of the study in accordance with the overall study goals when using students as participants. Additional studies comparing students and professionals under various empirical settings can be conducted to understand similarities and differences among students and professionals.

Some studies require participants from multiple demographics. For instance, S01 compares novices vs. experienced users in problem solving using patterns. In such cases, both demographic groups should be distinguishable and clear criteria for rating participants to determine expertise level should be established, as discussed in next section. Even when grouping of participants is not carried out based on experience level, we still need to rate participants on experience level to create balanced groups and to assess applicability of the findings [50].

Students may be a suitable demographics based on study goals, nature of tasks, and work experience of students. For instance, if the study focuses on using software patterns in education and training, students provide a good sample. Variations among students in terms of work experience, knowledge of patterns, and knowledge of problem domain should also be factored in so that applicability and generalizability of the findings could be established.

5.2. Determination of expertise

Software patterns are principally aimed at improving performance in the construction of software. The primary studies included in our mapping study sought to attribute an effect of change in programming, design or communication based on pattern use (see Fig. 2). To measure this effect, however, the investigators must measure expertise to establish a baseline for participants, as we have observed (see Table 7). Investigators should also control for participant expertise to determine what effect the pattern use contributes to the quality of the result as either independent or in conjunction with experience level. However, the reliance on experts without a standard mechanism to measure expertise limits the comparability of results: 'expert' in one study might be considered at a 'medium' or 'novice' level in another study. Differences between experts and non-experts, in terms of practical experience and familiarity with the problem domain, might also be more pronounced in some cases than others. For example, experts in S14 were graduate students with at least one year of work experience whereas experts in S01-B were individuals who work on similar tasks as given in the experiment on a daily basis (simulation consultants). In both cases, non-experts were undergraduate students with minimal work experience and lack of familiarity with problem domain.

Whereas self-reporting number of years in a field is easier to measure by subtracting employment start and end dates, other factors such as KLOC written and self-report ranking is subject to skewed estimation and individual bias. Expertise can be decomposed into skills that can be separately measured using skill test batteries for cognitive load, working memory, and reading comprehension, among others, to more precisely attribute the effect to participant skill levels across the sampled population [39]. The 'Joint Replication Project' underscores the significance of providing context to assess the similarity among participants and findings across multiple experiments [30].

In addition, several studies employed experts to evaluate the extent to which an artifact exhibits quality, correctness and creativity. However, to lend credibility to the evaluation, we again need to establish what makes these evaluators experts in the given problem domain [51].

Establishing a standard measure for experience level and expertise of participants across studies can help in identifying the demographics to which the findings are applicable. However, experience is difficult to establish and is based on multiple factors including participants background as well as familiarity with the given problem domain, set of patterns, and set of tasks. As a start, studies should report all of the above factors in detail.

5.3. Standard problem sets

Throughout our mapping study, we observed that different studies employed different protocols for delivering problems for participants to work on. The choice of problem sets begins with training materials, where participants first learn about patterns through tutorials, workshops, and courses (see Table 8). In addition, investigators employ other problem sets to evaluate the participant's performance by asking him or her to apply patterns to those problems (see Table 11). However, we noticed that no particular tool or framework is used in more than one study except for JHotDraw application employed in three studies, all conducted by same group of researchers. Essentially, researchers used different problem sets, tools and frameworks to perform the given task, which limits the synthesis of findings across the studies.

Sim et al. [52] observe that benchmarks facilitate technological progress and community building within a research area. Researchers in a number of computer science fields, such as image processing, network traffic monitoring, and recently social network analysis, routinely use standard problem sets to compare and contrast the performance of different algorithms and solution methodologies. In software engineering Heckman et al. [53] have established a benchmark for evaluating static analysis alert prioritization and classification techniques. Software patterns research can also benefit from such standardization. In study S14, researchers investigated how participants performed on a maintenance task using two functionally equivalent program versions, an original program version without design patterns and a refactored program version with design patterns. The underlying assumption was that participants working on refactored version will use the patterns to carry out the maintenance task. However, based on a follow up analysis in study S16, the researchers identified that there may be pattern-unaware solutions in the refactored program version and the maintenance task could be carried out without modifying the patterns. Using standard problem sets can further help in understanding such confounding context variables that may exist within a program. Standard problem sets could be published with accompanying training materials, such as textual formats and example presentations of the patterns to be given to participants.

Researchers use different problem sets, tools and frameworks to perform a given task, which limits the synthesis of findings across the studies. To support comparative evaluation, investigators should develop a standard set of problems and training materials for the patterns of interest. Availability of a package containing problem sets, instructions, and training material used in a study, is found conducive for replication.

5.4. Usage of preparatory material

As evidenced by the studies we reviewed, material used to prepare participants for the task could range from textual, instructional (or in-class, interactive lecture plans) and pre-recorded

video formats. Subtle differences in the duration and kind of training may impact performance [54]. For instance, in study S12, the group with the least detailed material provided a third of the functionality compared to the group with most detailed material containing concrete task-specific examples in the material text and design.

Due to the qualitative nature of preparatory material, there is ample opportunity to introduce bias or preference towards a treatment condition. If an investigator is comparing pattern vs. non-pattern approaches, for example in S03, removing details from preparatory material for one approach can skew the results in favor of the other approach. Establishing the equivalence of information content across different sets of preparatory material, to avoid information gain or loss in one set over another, is essential when constructing multiple sets of preparatory material conveying same basic information. If difference in training is not a factor in study design, use of standardized training materials can lessen the impact of training on the observed effect in the study.

Finally, because participants can neglect preparatory material that was provided, either by skipping or skimming this material, investigators should measure participant comprehension of such materials to understand their effect on the study outcome.

Subtle differences in the duration and kind of training may impact performance. Establishing equivalence of different sets of preparatory material is important if difference in training is not a factor in study design. Moreover, when evaluating the effect of different training material on study outcomes, the provision of training material should be followed by a comprehension test to see if training had any effect on knowledge acquisition.

5.5. Study themes and evaluation constructs

Researchers have investigates various themes related to the use of software patterns and evaluated different constructs to determine the effect of pattern use on desirable outcomes. Based on the classification used earlier (see Table 4), we have identified how frequently each construct is used to evaluate the outcome for each theme in Fig. 4. When evaluating the use of software patterns in maintenance, most frequently used constructs are efficiency, correctness and completeness respectively. These are also the most commonly used constructs for evaluation in general (see Fig. 3). Usability, in terms of identifying applicable patterns and correctly instantiating patterns in the context of the problem, is also used frequently as an evaluation criteria for studies related to the themes of software construction and performance. As one would expect, communicability during problem solving is the most commonly used construct for studies belonging to the theme of communication. Only time when efficiency is not one of the most frequently used criteria is while evaluating the use of software patterns during software construction.

Based on our analysis, time to complete a task can serve as a suitable measure for efficiency. However, when time is self-reported by participants, the measurement might not be as reliable. If incomplete or erroneous solutions are present, we also need to account for time needed to complete the task or to correct the errors. Measures for efficiency, completeness and correctness can be used in conjunction to have a realistic assessment of participants' performance. 'Learning efficiency', as self-reported by participants is also used to measure efficiency. Although self-reported qualitative measures may not be reliable, they can provide useful insights into the problem solving process and can be used to generate hypotheses for further evaluation. In a number of studies, researchers used qualitative measures such as quality, communicability or usability as evaluated by subject matter experts. When using qualitative measures, minimizing subjectivity is an important consideration. Researchers should consider using multiple evaluators and report inter-rater agreement between evaluators. When oracles, checklists or 'gold standards' are used to evaluate the outcome, researchers should describe mechanisms used to create these artifacts to lend validity to the findings of the study [37].

Often a construct that is being explored in the study is not well aligned with the measurement taken. For instance, studies S01 (A, B, C) assessed whether pattern use improves domain knowledge and design skills, whereas the variables measured primarily concern efficiency (time to complete, and ease of performing the task) and not knowledge acquisition. Because design skill is highly subjective and there are no established metrics, this is a reasonably difficult construct to measure and convenient proxy variables are easily mistaken for the construct. Such constructs may be

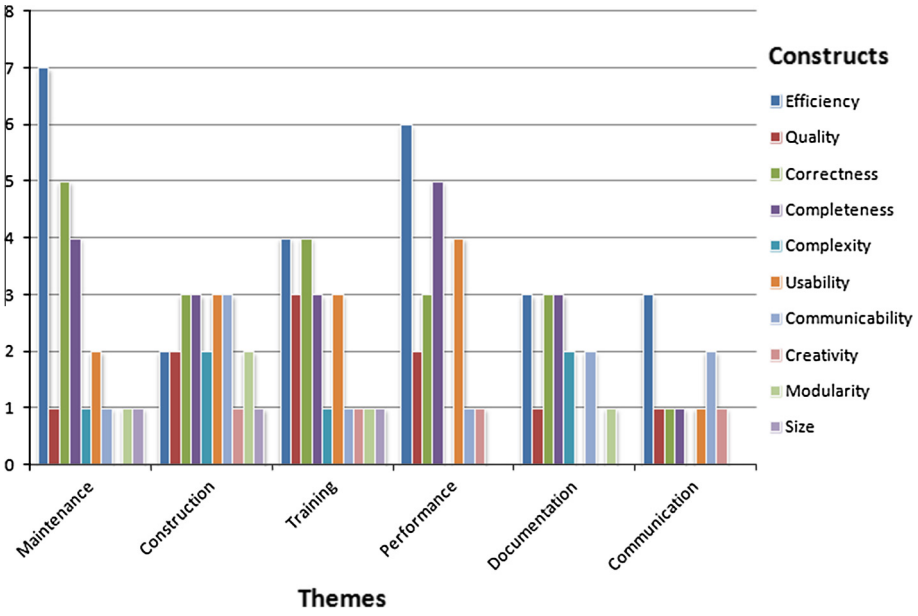


Fig. 4. Frequencies of constructs used in the primary studies for each theme.

qualitatively evaluated by subject matter experts. Research is ongoing to understand the impact of software patterns on various quality attributes [24]. In general, software engineering investigators needs to conduct more fundamental research on which measures correspond to which independent and dependent variables and under what conditions those measures are both reliable and valid.

Measures for efficiency, completeness and correctness can be used in conjunction to have a realistic assessment of participants' performance. Self-reported qualitative measures may not be reliable but can provide useful insights into the problem solving process. For constructs that are difficult to measure and have no established metrics, such as design quality, multiple subject matter experts may be employed to evaluate the outcome, preferably using pre-specified quality checklists and reporting inter-rater agreement.

6. Threats to validity of our mapping study

We now discuss the threats to validity of our mapping study.

Selection *of search terms and digital libraries:* Selection of search terms and digital libraries can preclude some relevant studies from appearing in our search results. We searched four digital libraries and two journals that tend to cover high quality publications in software engineering. We have also looked at the references of 57 papers, shortlisted after exclusion based on title and abstracts, to identify additional studies that we might have missed during primary search. However, studies published in proceedings not indexed by the four digital libraries may not appear in our analysis.

Selection *of studies:* We are interested in empirical evaluation of software pattern application involving human participants. Our selection criteria would exclude studies that consider a software artifact in isolation or focus on structural or quality aspect of patterns without involving pattern application. We would also exclude surveys where participants provide their opinion about software patterns without using them in problem solving in an empirical setting. Our findings should be considered applicable only in the context of the selected studies. We have also defined a criteria for quality assessment of included studies based on the aims and research questions of our study. We did not exclude any study based on the quality assessment results however the quality of included studies is fairly uniform.

Reliability *of data extraction and categorization:* We extracted and categorized data from the included studies to answer our research questions. We have identified each category by carrying out a constant comparison of instances of data across all studies in accordance with grounded analysis practices. We report all identified categories without omission in our analysis. The extraction and categorization process was carried out by the first author, a graduate student with over five years of work experience in software engineering and design patterns. The first two co-authors provided input to resolve ambiguities during the process. In this respect, the extraction and categorization process is partially validated.

7. Conclusion

In this paper, we have reported the results of our systematic mapping study documenting and analyzing the current state of empirical research on the use of software patterns in problem solving. While analyzing structural aspects of patterns in isolation can provide insights into the inherent properties of a pattern, use of patterns during problem solving involves additional context factors such as participants, problems, domains and time constraints. A pattern that has good quality attributes may not be useful during problem solving if not instantiated properly in the context of the problem being solved [55]. Characterization of existing empirical research, evaluating the use of software patterns by human participants, is a first step towards understanding the dynamics of pattern application in problem solving.

We selected 30 primary studies that evaluate design patterns, software architecture patterns and security patterns, among others. Five of the included studies report case studies while the remaining are experiments. We classified the primary studies in terms of their research questions, empirical design context factors, measures, and threats to validity. Despite the availability of guidelines for conducting empirical studies in software engineering [46], we found gaps in adherence to the guidelines. For instance, when using qualitative measures, no detail on minimizing biases and subjectivity in assessment is provided 27% of the time. Moreover, 9 of the primary studies do not discuss any threats to validity of the study design and execution. However, the quality score for studies published in recent years is consistently high indicating that the researchers are providing more details in terms of study design and execution.

We found that while evaluating the problem solving process, measures such as 'time to complete a task' and 'learning efficiency' are consistently used across studies. This might indicate that empirical evaluation of software patterns is converging towards a set of standard measures to evaluate how software patterns support problem solving process. However we did not identify measures that are consistently used to evaluate the final artifact itself even though majority of the measures are related to the artifact. Since we are only considering studies involving human participants, studies evaluating the final artifact that do not involve human participants are not included in our analysis. An analysis of such studies might lead to identification of commonly used measures for evaluating the final artifact. Different problem domains, patterns and tasks might account for the use of different measures for evaluating the final artifact as well. Two-thirds of the primary studies have explicitly considered threats to validity of study design. While internal and external validity is frequently assessed, researchers are increasingly considering construct validity as well as validity of the conclusions drawn as a result of the study.

We identified that subtle differences in study design can significantly limit comparison of experimental results across studies. For instance, despite the common use of participant categories as either novices or experts, we found that we could not compare these categories among different studies. Each study uses a different criteria to measure expertise based on the problem domain and relative training in solving the given problem since expertise is not independent of the pattern and problem domains in such experiments. Observations based on experience level of participants, without establishing a standard definition for experience level, can be difficult to generalize. These issues can affect the ability to compare results across studies, which sample from different demographics. As software engineering continues to borrow case study and experimental design practices from other disciplines, such as sociology and psychology, more work is needed to adapt these methods to software engineering and streamline practices to the point that quantitative comparisons can be made. In addition, establishing baselines in terms of the experience level of participants, the problem sets participants work on, and the measures used to evaluate participants' performance can support replication and comparison of findings across studies.

The results of our mapping study highlight the current state of empirical research on software pattern application in terms of design, measurements and use in problem solving. Based on our mapping study, we have identified commonly employed empirical

protocols, the level of details in reporting various context factors, and commonly used measures. Future researchers can leverage this information when designing studies while considering various threats to validity as reported herein. By using common measures and protocols, researchers can design new studies, replicate existing studies and be able to compare the findings. However, we observed that a number of context factors limit comparability of the findings. We have discussed these aspects in the context of our mapping study. Further evaluation of software patterns in empirical settings is needed to explore and address these issues by emphasizing reliability, repeatability and comparability of the studies.

Acknowledgments

We thank the NCSU Realsearch Group and the CMU Requirements Engineering Lab for their valuable feedback on this paper. We give special thanks to Ben Smith and Jason King for their help with voting on the papers. This research was funded by the Army Research Office (W911NF-09-1-0273).

Appendix A. List of selected studies

Reference code	Study reference
S01 (A, B, C)	G. Kolfschoten, S. Lukosch, A. Verbraeck, and E. Valentin, "Cognitive learning efficiency through the use of design patterns in teaching," *Computers and Education,* vol. 54, pp. 652–660, April 2010
S02	A. Chatzigeorgiou, N. Tsantalis, and I. Deligiannis, "An empirical study on students' ability to comprehend design patterns," *Computers and Education,* vol. 51, pp. 1007–1016, 2008
S03 (A, B)	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 Transactions on Software Engineering,* vol. 28, 2002
S03-R1	M. Torchiano, "Documenting pattern use in java programs," in *International Conference on Software Maintenance (ICSM'02)*, Montreal, Canada, 2002, pp. 230–233
S04	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 Transactions on Software Engineering,* vol. 27, pp. 1134–1144, 2001
S04-R1	M. Vocak, W. F. Tichy, D. I. K. Sjøberg, E. Arisolm, 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,* vol. 9, pp. 149–195, Sept 2004
S04-R2	L. Prechelt and M. Liesenberg, "Design patterns in software maintenance: An experiment replication at Freie Universiẗat Berlin," in *Second International Workshop on Replication in Empirical Software Engineering Research (RESER'11)* 2011, pp. 1–6
S04-R3	N. Juristo and S. Vegas, "Design patterns in software maintenance: An experiment replication at UPM - Experiences with the RESER'11 Joint Replication Project," in *Second International Workshop on Replication in Empirical Software Engineering Research (RESER'11)*, 2011, pp. 7–14
S04-R4	A. Nanthaamornphong and J. C. Carver, "Design patterns in software maintenance: An experiment replication at University of Alabama," in *Second International Workshop on Replication in Empirical Software Engineering Research (RESER'11)*, 2011, pp. 15–24
S04-R5	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," in *Second International Workshop on Replication in Empirical Software Engineering Research (RESER'11)* 2011, pp. 25–34
S05	M. A. Jalil and S. A. M. Noah, "The difficulties of using design patterns among novices: An Exploratory Study," in *5th International Conference on Computational Science & Applications (ICCSA'07)*, Kuala Lampur, Malaysia, 2007, pp. 97–103
S06	E. Chung, J. Hong, M. Prabaker, J. Landay, and A. Liu, "Development and evaluation of emerging design patterns for ubiquitous computing," in *Conference on Designing Interactive Systems (DIS'04)*, 2004, pp. 233–242
S07	B. Ellis, J. Stylos, and B. Myers, "The factory pattern in API design: A usability evaluation," in *29th International Conference on Software Engineering (ICSE'07)*, Minneapolis, MN 2007, pp. 302–311
S08	S. Jeanmart, Y. G. H. Sahraoui, and N. Habra, "Impact of the visitor pattern on program comprehension and maintenance," in *Third International Symposium on Empirical Software Engineering & Measurement (ESEM'09)*, Lake Buena Vista, FL, 2009, pp. 69–78
S09	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," in *International Workshop on Software Quality (WoSQ'06)*, 2006, pp. 51–56
S10	T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu, "Do maintainers utilize deployed design patterns effectively?," in *29th International Conference on Software Engineering (ICSE'07)*, Minneapolis, MN, USA, 2007
S11	B. Unger and W. Tichy, "Do design patterns improve communication? an experiment with pair design," in *International Workshop on Empirical Studies of Software Maintenance*, 2000, pp. 1–5
S12	E. Golden, B. E. John, and L. Bass, "The value of a usability-supporting architectural pattern in software architecture design: A controlled experiment," in *27th International Conference on Software Engineering (ICSE'05)*, St. Louis, MI, 2005
S13	R. Porter and P. Calder, "Patterns in learning to program - An experiment?," presented at the *Sixth Australasian Conference on Computing Education* (ACE'04), 2004
S14	T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu, "Work experience vs. refactoring to design patterns: A controlled experiment," in *14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. (SIGSOFT'06/FSE-14)* Portland, OR, 2006, pp. 12–22

Appendix A (*continued*)

Reference code	Study reference
S15	T. Ihme and P. Abrahamsson, "The use of architectural patterns in the agile software development of mobile applications," in *International Conference on Agility (ICAM'05)*, Helsinki, Finland, 2005
S16	T. H. Ng, Y. T. Yu, S. C. Cheung, and W. K. Chan, "Human and program factors affecting the maintenance of programs with deployed design patterns," *Information and Software Technology,* vol. 54, pp. 99–118, 2012
S17	U. v. Heesch, P. Avgeriou, U. Zdun, and N. Harrison, "The supportive effect of patterns in architecture decision recovery—A controlled experiment," *Science of Computer Programming,* vol. 77, pp. 551–576, 2012
S18	K. Yskout, R. Scandariato, and W. Joosen, "Does organizing security patterns focus architectural choices?," in *International Conference on Software Engineering (ICSE '12)*, 2012
S19	J. Abramova, A. Sturma, and P. Shovala, "Evaluation of the Pattern-based method for Secure Development (PbSD): A controlled experiment," *Information and Software Technology,* vol. 54, pp. 1029–1043, 2012
S20	A. W. Kamal, P. Avgeriou, and U. Zdun, "The use of pattern participants relationships for integrating patterns: a controlled experiment," *Software - Practice and Experience*, vol. 43, pp. 807–833, 2013
S21	G. Mangalaraj, S. Nerur, R. Mahapatra, and K. H. Price, "Distributed cognition in software design: An experimental investigation of the role of design patterns and collaboration," *Management Information Systems Quarterly*, vol. 38, pp. 249–274, 2014

Appendix B. Quality appraisal of included studies

Categories: R = Research questions/Hypothesis specified; **S** = Sample described; **P** = Patterns details provided; **F** = Findings reported; **L** = Limitations/Threats to validity discussed; **Quality Assessment Score: 0** = Not at all; **1** = Somewhat; **2** = Mostly; **3** = Fully**;**

Reference code	Publication venue	Year	# of pages	Quality assessment score					
				R	S	P	F	L	Average (0–3)
S01 (A, B, C)	Computers and Education, Journal	2010	9	3	3	3	3	1	2.6
S02	Computers and Education, Journal	2008	10	2	2	3	3	3	2.6
S03 (A, B)	IEEE Transactions on Software Engineering, Journal	2002	10	3	3	3	3	3	3
S03-R1	International Conference on Software Maintenance	2002	4	3	2	3	1	0	1.8
S04	IEEE Transactions on Software Engineering, Journal	2001	11	3	3	3	3	3	3
S04-R1	Empirical Software Engineering, Journal	2004	48	3	3	3	3	3	3
S04-R2	Int'l Workshop on Replication in Empirical Software Engineering Research (RESER)	2011	6	3	3	3	2	3	2.8
S04-R3	RESER	2011	8	3	1	3	3	3	2.6
S04-R4	RESER	2011	10	3	3	3	3	3	3
S04-R5	RESER	2011	10	3	3	3	3	3	3
S05	International Conference on Computational Science & Applications	2007	7	2	2	3	3	0	2
S06	Conference on Designing Interactive Systems	2004	10	3	3	3	3	1	2.6
S07	International Conference on Software Engineering (ICSE)	2007	10	2	3	3	3	3	2.8
S08	Empirical Software Engineering and Measurement	2009	10	3	3	3	3	3	3
S09	International Workshop on Software Quality	2006	6	3	3	3	3	3	3
S10	ICSE	2007	10	3	2	3	3	3	2.8
S11	International Workshop on Empirical Studies of Software Maintenance	2000	5	3	3	3	2	3	2.8
S12	ICSE	2005	10	2	3	3	3	1	2.4
S13	Australasian Conference on Computing Education	2004	6	2	1	2	2	1	1.6
S14	International Symposium on Foundations of Software Engineering	2006	11	3	2	3	3	3	2.8
S15	International Conference on Agility	2005	9	3	2	3	3	1	2.4
S16	Information and Software Technology, Journal	2012	29	3	2	3	3	3	2.8
S17	Science of Computer Programming, Journal	2012	26	3	3	3	3	3	3
S18	ICSE	2012	11	3	3	3	3	3	3
S19	Information and Software Technology, Journal	2012	15	2	2	3	3	3	2.6
S20	Software – Practice and Experience	2013	27	3	2	3	3	3	2.8
S21	Management Information Systems Quarterly	2014	32	3	3	3	3	3	3

Appendix C. Research questions and hypotheses of included studies

Reference Code	Research Questions and Hypotheses of Included Studies (RQs that are inferred are in *italics*)
S01 (A, B, C)	• H_{a1}: Novices will faster gain understanding in problem solving and design skills, when they learn to design with the design patterns approach first, before they learn to understand entire systems
	• H_{a2}: Experienced (domain experts or more experienced designers) will not experience a learning effect from the use of design patterns, but might find them useful in other ways
	• R_1: Training novices with the use of design patterns will increase the quality of the schemas they build to represent a system
S02	• R_1: *During pattern selection, to what extent are students able to identify applicable design patterns?*
	• R_2: *During pattern application, to what extent are students able to comprehend and instantiate design patterns?*
S03 (A, B); S03-R1	• R_1: Does it help the maintainer if the design patterns in the program code are documented explicitly (using source code comments) compared to a well-commented program without explicit reference to design patterns?
	• H_{11}: By adding pattern comment lines (PCL), pattern-relevant maintenance tasks are completed faster
	• H_{12}: By adding PCL, fewer errors are committed in pattern-relevant maintenance tasks
S04; S04-R1	• H_{01}: Design pattern P does not improve the performance of subjects doing a maintenance
***through* S04-R5**	• exercise X on program A (containing P) when compared to subjects doing the same exercise X on an alternative program A' (not containing P)
	• R_1: Do design patterns improve the understandability of software designs? *[S04-R4]*
S05	• R_1: *What are the problems associated with pattern usage among novices?*
S06	• R_2: Are patterns useful for introducing designers to ubicomp?
	• R_3: Are patterns useful for communicating between designers? For example, do designers adopt the pattern language vocabulary as they talk about a design?
	• R_4: Are patterns useful for creating designs?
	• R_5: Are patterns useful for creating higher-quality designs?
S07	• R_1: *How does the Factory pattern compare to using constructors in API design?*
S08	• H_{11}: A class diagram with the Visitor reduces the subjects' efforts during program comprehension when compared to one without it
	• H_{12}: A class diagram using the canonical representation of the Visitor reduces the subjects' efforts during program comprehension when compared to one using the Visitor and another layout
	• H_{21}: A class diagram with the Visitor reduces the subjects' efforts during program modification when compared to a class diagram without it
	• H_{22}: A class diagram using the canonical representation of the Visitor reduces the subjects' efforts during program modification when compared to one using the Visitor and another layout
S09	• H_{01}: When the theme of patterns is not satisfied, there is at least 20% of chance where the Open-Closed principle is followed
	• H_{02}: When the theme of patterns is satisfied, there is at least 20% of chance where the Open-Closed principle is violated
S10	• R_1: What proportion of maintainers utilize the relevant design patterns in completing the required anticipated change?
	• R_2: Do maintainers have a higher tendency to perform one task over another?
	• R_3: Comparing the codes delivered by maintainers who utilize the relevant design patterns in completing the required anticipated change to those delivered by maintainers who do not utilize patterns, are they equally faulty?
S11	• H_{11}: If team members have common design pattern knowledge and vocabulary, they can communicate more effectively than without
S12	• R_1: *Do the Usability-Supporting Architecture Patterns (USAP) help in considering relevant responsibilities and usability concerns during architectural design?*
S13	• R_1: How do patterns help in learning to program?
S14	• R_1: Would refactoring a program using design patterns (in)conclusively supersede the effect of work experience to guide maintainers to complete a maintenance task, or vice versa?
S15	• R_1: How to select and present patterns in the training material so as to motivate and facilitate their use by the project team?
	• R_2: How can patterns help the team to create new software effectively, while simultaneously maintaining the quality of the developed software at a high level?
	• R_3: How can patterns help the team to develop useful and effective software architectures?
	• R_4: Are the Mobile-D approach with its agile values and the Agile Architecture Line Model suitable to be used with patterns?
	• R_5: How can patterns be used as a documentation aid to preserve vital design information that helps to maintain and evolve the developed product after the project?

Appendix C (*continued*)

Reference Code	Research Questions and Hypotheses of Included Studies (RQs that are inferred are in *italics*)
S16	• R_1: What human and program factors may contribute to the productivity (in terms of time spent) of maintainers in making correct software changes when they work on programs with deployed design patterns (Factors: Deployment of design patterns; Presence of pattern-unaware solutions; Prior exposure to design patterns; Prior exposure to the program; Prior exposure to the programming language; Prior work experience)?
S17	• H_{11}: The quality of recovered decisions is higher when the recovery focuses on identifying patterns in the architecture, compared to ad-hoc, intuitive recovery • H_{12}: The quantity of recovered decisions is higher when the recovery focuses on identifying patterns in the architecture, compared to ad-hoc, intuitive recovery
S18	• R_1: Does Organizing Security Patterns Focus Architectural Choices?
S19	• R_1: *How does the use of security patterns affect the quality of resulting database access control specifications when compared to coding the access control specifications directly?*
S20	• H_{00}: The use of pattern participants relationships does not help software architects to more accurately combine architectural patterns within software architectures as compared with integrating architectural patterns without using such relationships • H_{01}: The integration of architectural patterns using pattern participants relationships does not result in a more comprehensible software architecture as compared with integrating patterns without the use of such relationships • H_{02}: The use of pattern participants relationships does not help software architects to better document architectural design decisions as compared with documenting design decisions without using such relationships • H_{03}: The use of pattern participants relationships does not help software architects to more effectively partition a software architecture into components and sub-components, and assign responsibilities as compared with partitioning a software architecture without the use of such relationships
S21	• H_1: Quality of the design solution will be higher when design patterns are used during software design • H_2: Time taken to complete a design task will be shorter when design patterns are used • H_3: Solution quality of a collaborating pair will be higher than that of the second-best member of a nominal pair in a software design task • H_{4a}: Time taken by collaborating pairs to complete a software design task will be longer than that of the best member of a nominal pair • H_{4b}: Time taken by collaborating pairs to complete a software design task will be longer than that of the second-best member of a nominal pair • H_{5a}: When design patterns are available, the performance gap in solution quality between the second best member of a nominal pair and the best member of the nominal pair and the pair will be reduced in contrast to when design patterns are not used • H_{5b}: Solution quality of the second-best member of a nominal pair with design patterns will be higher than that of the collaborating pair without design patterns in a software design task • H_{5c}: Task completion time of the second-best member of a nominal pair with design patterns will be lower than that of the collaborating pairs without design patterns in a software design task • H_6: Task satisfaction will be higher when design patterns are used in a software design task • H_7: The average level of task satisfaction among collaborating pairs will be higher when compared with the average level of task satisfaction of nominal pairs

References

[1] P.N. Robillard, The role of knowledge in software development, in: Communications of the ACM, ACM New York, NY, USA, 1999, pp. 87–92.
[2] N. Charness, M. Tuffiash, The role of expertise research and human factors in capturing, explaining, and producing superior performance, Hum. Fact.: J. Hum. Fact. Ergon. Soc. 50 (3) (2008) 427–432.
[3] D.C. Schmidt, M. Fayad, R.E. Johnson, Software patterns, Commun. ACM 39 (10) (1996).
[4] D. Sjøberg, T. Dybå, M. Jørgensen, The future of empirical methods in software engineering research, in: Future of Software Engineering (FOSE'07), IEEE, 2007.
[5] B.A. Kitchenham, D. Budgen, O.P. Brereton, Using mapping studies as the basis for further research—a participant-observer case study, Inf. Softw. Technol., Spec. Sec. EASE 53 (4) (2011) 638–651.
[6] C. Alexander et al., A Pattern Language, Oxford University Press, New York, 1977.
[7] E. Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.
[8] M. Fowler, Analysis Patterns: Reusable Object Models, in: G.J. Booch Ivar, Rumbaugh James (Eds.), Object Technology Series, Addison Wesley Longman Inc, Menlo Park, CA, 1997.
[9] P. Clements, L. Northrup, Software Product Lines: Practices and Patterns, Addison-Wesley Professional, Reading, PA, 2001.
[10] M. Schumacher et al., Security Patterns: Integrating Security and Systems Engineering, John Wiley & Sons Ltd., West Sussex, 2006.
[11] D. C. Schmidt, M. Fayad, R.E. Johnson, Software patterns, in: Communications of the ACM, 1996.
[12] G. Cooper, J. Sweller, Effects of schema acquisition and rule automation on mathematical problem-solving transfer, J. Educ. Psychol. 79 (4) (1987) 347–362.
[13] C. Kohls, K. Scheiter, The relation between design patterns and schema theory, in: 15th Conference on Pattern Languages of Programs, 2008, pp. 1–14.
[14] M. Jackson, Problem frames and software engineering, Inf. Softw. Technol. 47 (14) (2005) 903–912.
[15] M. Jackson, Problem Frames, Addison-Wesley, 2001.
[16] D. Budgen, et al., Using mapping studies in software engineering, in: 20th Annual Psychology of Programming Interest Group Conference (PPIG 2008), Lancaster University, UK, 2008.
[17] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE-2007-01 School of Computer Science and Mathematics, Keele University, 2007.
[18] F.Q.B.d Silva, et al., A critical appraisal of systematic reviews in software engineering from the perspective of the research questions asked in the reviews, in: Empirical Software Engineering and Measurement (ESEM), Bolzano-Bozen, Italy, 2010.
[19] R.M. Lindsay, A.S.C. Ehrenberg, The design of replicated studies, Am. Statist. 47 (3) (1993) 12.

[20] O.S. Gómez, N. Juristo, S. Vegas, Replication, Reproduction and Re-analysis: three ways for verifying experimental findings, in: 1st International Workshop on Replication in Empirical Software Engineering Research, RESER, ACM, Cape Town, South Africa, 2010.
[21] C. Zhang, D. Budgen, What do we know about the effectiveness of software design patterns?, IEEE Trans Softw. Eng. 38 (5) (2012) 1213–1231.
[22] A. Ampatzoglou, S. Charalampidou, I. Stamelos, Research state of the art on GoF design patterns: a mapping study, J. Syst. Softw. 86 (7) (2013) 1945–1964.
[23] M. Ali, M.O. Elish, A comparative literature survey of design patterns impact on software quality, in: International Conference on Information Science and Applications (ICISA), Suwon, 2013.
[24] M. Galster, P. Avgeriou, Qualitative analysis of the impact of SOA patterns on quality attributes, in: 12th International Conference on Quality Software, 2012.
[25] K. Petersen, et al., Systematic mapping studies in software engineering, in: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08), 2008.
[26] A. Ampatzoglou et al., An empirical investigation on the reusability of design patterns and software packages, J. Syst. Softw. 84 (2011) 2265–2283.
[27] F. Khomh, Y.-G. Guéhéneuc, Do design patterns impact software quality positively? in: 12th European Conference on Software Maintenance and Reengineering, 2008.
[28] H. Zhang, M.A. Babar, On searching relevant studies in software engineering, in: 14th International Conference on Evaluation and Assessment in Software Engineering, 2010.
[29] O. Dieste, A. Grimán, N. Juristo, Developing search strategies for detecting relevant experiments, Emp. Softw. Eng. 14 (2009) 513–539.
[30] L. Prechelt, M. Liesenberg, Design patterns in software maintenance. An experiment replication at Freie Universiïat Berlin, in: Second International Workshop on Replication in Empirical Software Engineering Research, 2011.
[31] B. Kitchenham, et al., Trends in the quality of human-centric software engineering experiments – a quasi-experiment, IEEE Trans. Softw. Eng. (2012).
[32] S. Adolph, W. Hall, P. Kruchten, A methodological leg to stand on: lessons learned using grounded theory to study software development, in: CASCON2008, pp. 166–178.
[33] C. Wohlin, et al., in: V.R. Basili (Ed.), Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Sweden, 2000.
[34] J.W. Creswell, Research Design: Qualitative, Quantitative and Mixed Methods Approaches, second ed., Sage Publications, 2003.
[35] W. Shadish, T. Cook, D. Campbell, Experimental and Quasi Experimental Designs for Generilized Causal Inference, Houghton Mifflin, Boston, 2002.
[36] V. Basili, Software Modeling and Measurement: The Goal/Question/Metric Paradigm, University of Maryland, 1992.
[37] R.K. Yin, Case Study Research: Design and Methods, forth ed., Sage Publications, 2009.
[38] D.M. Lane, Research design, in: Online Statistics Education: An Interactive Multimedia Course of Study, Rice University.
[39] J. Shanteaua et al., Performance-based assessment of expertise: how to decide if someone is an expert or not, Eur. J. Oper. Res. 136 (2) (2002) 11.
[40] R.O. Briggs, et al., ThinkLets: achieving predictable, repeatable patterns of group interaction with group support systems (GSS), in: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001.
[41] W.D. Kelton, R.P. Sadowski, D.A. Sadowki, Simulation with Arena, second ed., McGraw-Hill, 2002.
[42] T. Schümmer, S. Lukosch, Patterns for Computer-Mediated Interaction, John Wiley & Sons Ltd., Chichester, England, 2007.
[43] M. Vokac, Defect frequency and design patterns: an empirical study of industrial code, IEEE Trans. Softw. Eng. 30 (12) (2004) 904–917.
[44] V. Balijepally et al., Are two heads better than one for software development? The productivity paradox of pair programming, Manage. Inf. Syst. Quart. 32 (4) (2009) 91–118.
[45] A.J. Viera, J.M. Garrett, Understanding interobserver agreement: the kappa statistic, Fam. Med. 37 (5) (2005) 360–363.
[46] B. Kitchenham et al., Evaluating guidelines for reporting empirical software engineering studies, Emp. Softw. Eng. 13 (2008) 97–121.
[47] M. Höst, B. Regnell, C. Wohlin, Using students as subjects—a comparative study of students and professionals in lead-time impact assessment, Emp. Softw. Eng. 5 (3) (2000) 201–214.
[48] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, Lawrence Erlbaum Associates, Hillsdale, NJ, 1988.
[49] J.C. Carver et al., A checklist for integrating student empirical studies with research and teaching goals, Emp. Softw. Eng. 15 (2010) 35–59.
[50] S. Kleinschmager, S. Hanenberg, How to rate programming skills in programming experiments? A preliminary, exploratory, study based on university marks, pretests, and self-estimation, in: 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU '11), 2011.
[51] H. Hibshi, et al., Towards a framework to measure security expertise in requirements analysis, in: 1st International Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE'14), Sweden, 2014.
[52] S.E. Sim, S. Easterbrook, R.C. Holt, Using benchmarking to advance research: a challenge to software engineering, in: 25th International Conference on Software Engineering, 2003.
[53] S. Heckman, L. Williams, On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques, in: 2nd International Conference on Empirical Software Engineering and Measurement (ESEM 2008), Kaiserslauten, Germany, 2008.
[54] A. Rao, et al., Investigating the role of examples in security studies using analogical transfer, in: 1st ACM Symposium and Bootcamp on the Science of Security (HOTSOS'14), Raleigh, NC, 2014.
[55] M.A. Jalil, S.A.M. Noah, The difficulties of using design patterns among novices: an exploratory study, in: 5th International Conference on Computational Science & Applications (ICCSA'07), Kuala Lampur, Malaysia, 2007.