On the Design of Empirical Studies to Evaluate Software Patterns: A Survey

Maria Riaz
North Carolina State University
Dept. of Computer Science
Raleigh, NC, USA
mriaz@ncsu.edu

Laurie Williams
North Carolina State University
Dept. of Computer Science
Raleigh, NC, USA
williams@csc.ncsu.edu

Travis Breaux
Carnegie Mellon University
Institute for Software Research
Pittsburgh, Pennsylvania, USA
breaux@cs.cmu.edu

Jianwei Niu
University of Texas – San Antonio
Department of Computer Science
San Antonio, Texas, USA
niu@cs.utsa.edu

ABSTRACT
Software patterns are created with the goal of capturing expert knowledge so it can be efficiently and effectively shared with the software development community. However, patterns in practice may or may not achieve these goals. Empirical studies of the use of software patterns can help in providing deeper insight into whether these goals have been met. The objective of this paper is to aid researchers in designing empirical studies of software patterns by summarizing the study designs of software patterns available in the literature. The important components of these study designs include the evaluation criteria and how the patterns are presented to study participants. We select and analyze 19 distinct empirical studies and identify 17 independent variables in three different categories (participants demographics; pattern presentation; problem presentation). We also extract 10 evaluation criteria with 23 associated observable measures. Additionally, by synthesizing the reported observations, we identify challenges faced during study execution. Provision of multiple domain-specific examples of pattern application and tool support to assist in pattern selection are helpful for the study participants in understanding and completing the study task. Capturing data regarding the cognitive processes of participants can provide insights into the findings of the study.

Categories and Subject Descriptors
Architecture and design, Empirical studies of software engineering, Patterns and frameworks.

General Terms
Documentation, Design, Experimentation, Human Factors.

Keywords
Software patterns, experiment design, empirical evaluation, knowledge transfer

ACM SIGSOFT'12, FSE-20, November 10–17, 2012, Research Triangle Park, North Carolina, USA.

1. INTRODUCTION
Software has become increasingly pervasive in modern information systems, from control systems in transportation and energy to mobile apps that pull data into a user's personal context. While the availability of frameworks and tools enable developers to build more complex systems by reusing software components, designing reliable software continues to depend on the availability of expert knowledge [1]. Software design patterns [13] are one approach for capturing and sharing expert knowledge. Design patterns are generic, reusable structures that a software developer can "instantiate" into their system design to solve a problem. The Visitor Pattern, for example, allows developers to elegantly compartmentalize "processing code" for a collection of objects from different classes to improve maintainability [13]. Architectural patterns in building design [2] inspired design patterns in software engineering. The success of design patterns then inspired many other software patterns, such as enterprise application architectural patterns [10] and software product line patterns [8]. But, how can we know the software patterns meet their goal of efficiently and effectively sharing expert knowledge to the software development community such that the community is aided by the patterns?

Despite the intuitive appeal of pattern-based design, software engineering researchers have sought to empirically evaluate design pattern use in experimental settings in addition to qualitative appraisal [26]. Empirical evidence of the benefits of the use of a type of software patterns may increase the use of the patterns in the community.

Pattern discovery is primarily an expert's affair: repetitive solutions to slightly dissimilar problems leads to the expert's creative insight of a common, generalizable pattern. This skillset relies on the Revised Bloom Taxonomy's highest level of learning (analyze and create) [31]. An advantage of design patterns is that a less experienced designer can reuse the pattern, thus appealing to lower, more pervasive learning levels (remember and understand) [31]. The objective of this paper to aid researchers in designing empirical studies of software patterns by providing: (1) *evaluation criteria* and *observation measures* used to empirically assess the efficiency and effectiveness of software design patterns in imparting expert knowledge; and (2) pattern *presentation attributes* that affect (increase or decrease) this effectiveness. To this end, we report our results from a software engineering

literature survey on the topic of the empirical assessment of software design patterns.

The software patterns community has identified several assertions to promote the value of patterns to designers, such as: facilitation and ease of reuse; identification and capture of abstract concepts; aide in defining interfaces and interactions; means of shared documentation; construction of software with defined properties; and provision of common vocabulary [4]. Experimental evaluation can provide a scientific basis for verifying or refuting these assertions. However, multiple parameters effect the experiment design including factors such as: level of participants' expertise in applying software patterns; the manner of pattern documentation and presentation to participants in an empirical setting; the evaluation criteria used to assess quality of the outcome (a software artifact) after applying the patterns; and the method used for empirical evaluation. Our literature survey reported herein to identify these parameters can be used by software design patterns researchers and developers as an experimental design roadmap when evaluating patterns for efficiency and effectiveness.

The remainder of our paper is organized as follows: In Section 2, we provide a brief background of different areas in which software patterns are used. We document our methodology for conducting the survey in Section 3. In Section 4, we present the results of the survey to answer our research questions. In Section 5, we discuss and analyze the results and present our observations.

2. BACKGROUND

Schmidt et al. [26] defines software pattern as a means of providing successful solutions to common software problems. Software patterns are created with the goal of capturing expert knowledge so it can be efficiently and effectively shared with the software development community. Design patterns [13] provide successful solutions to recurring problems in the context of a software design. In addition to design, concept of software patterns is used in the requirements [26], analysis [11], architecture [12] and configuration managements [12] among others. We focus on the empirical evaluation of software design patterns in different domains for the purpose of this literature survey.

3. METHODOLOGY

We conducted a literature survey following the principles of evidence-based software engineering, as discussed in [4]. In this section, we describe our methodology for conducting the literature survey.

3.1 Research Questions

We established the following research questions to focus our analysis of the studies in our survey.

RQ1. What are the demographics, including skill levels, of study participants involved in the study?

RQ2. How are the software patterns documented and presented to the participants?

RQ3. How are the problems (to be solved using the patterns) selected, documented and presented to the participants?

RQ4. What are the evaluation criteria used in the studies?

RQ5. What are the observable measures collected in the studies?

3.2 Inclusion and exclusion criteria

The inclusion criteria determines which papers are studied to answer our research questions; included papers must be:

1) A peer reviewed publication, in the field of software engineering, evaluating one or more criteria of *design pattern application* based on:
   a) Experimental studies involving human participants
   b) Case studies involving human participants
2) A peer reviewed publication, in the field of software engineering, evaluating one or more criteria of *software pattern* application based on:
   a) Experimental studies involving human participants
   b) Case studies involving human participants

We excluded the following papers:

1) Publications not related to computer science and software engineering
2) A peer reviewed publication, in the field of software engineering discussing:
   a) Design patterns in general without empirical evaluation
   b) Design pattern quality assessment in isolation, without empirical evaluation involving human participants
   c) Questionnaire-based assessment of design patterns
   d) Design pattern mining
   e) Design pattern recovery / discovery
   c) Reverse engineering using patterns
3) Books and non-peer reviewed publications on design patterns
4) Design pattern catalogs

3.3 Data Collection

We only include papers that report the following data items to describe their empirical methodology; we did not use these data items to evaluate study quality.

- Complete reference of the source publication
- Empirical method (case study, controlled experiment)
- Pattern presentation technique to the participants (tool, library, tutorial, introduction, instantiated patterns)
- Problem presentation technique (type of problem, size of problem, number of tasks, duration to complete, deliverable)
- Problem domain and specific development environment, if any
- Participants (number, expertise level, grouping)
- Type of evaluators (academic experts, industry experts)
- Evaluation criteria (see Table 5, below).
- Effect of using patterns (positive / negative) on participants
- Results discussion provided by publication authors

3.4 Search Process

We began an exploratory search phase to identify relevant terms to our literature survey. We searched three databases (Google Scholar, ACM, IEEE) and collected papers that had the search terms anywhere in the paper. We use the following search terms:

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

After eliminating duplicates, these search queries produced an initial set of 295 papers. We down-selected by having two researchers separately vote on the paper titles for their relevance to the empirical assessment of software patterns, erring on the side

of inclusion at this step. The combined union of these two sets yielded 95 papers. We further eliminated 64 papers by evaluating the paper abstracts based on the inclusion and exclusion criteria in Section 3.2. One researcher voted on all the abstracts while the other voted on a subset of 44 abstracts identified using the first two search terms. Any paper voted in favor by either of the researchers was included as before yielding a set of 31 papers.

One of these 31 papers was a mapping study conducted by Zhang and Budgen [31]. This paper discusses and summarizes 11 papers on experimental evaluation of the usefulness and usability of the 'Gang of Four' (or 'GoF' patterns) [13] design patterns. We applied our detailed inclusion criteria (in terms of data collection) on the set of 31 papers which yielded 14 papers. In addition, we included 11 papers from the mapping study: 8 of these papers were independently identified by our method. We added the three remaining papers from the mapping study to yield a total of 17 papers. All 11 papers discussed in the mapping study [31] are included in our survey as they satisfy our detailed inclusion criteria.

In our survey, we are interested in papers that provide empirical evaluation of software patterns including, but not limited to, the GoF design patterns. Mapping study by Zhang and Budgen [31] studied literature up to the end of 2009. We have surveyed literature published until January 2012 and include relevant papers published since the mapping study.

3.5 Included Papers

The list of included papers follows with a unique reference number (in brackets) used throughout the remainder of this paper:

[S01] Cognitive learning efficiency through the use of design patterns in teaching [18]
[S02] An empirical study on students' ability to comprehend design patterns [5]
[S03] Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance [25]
[S04] A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions [24]
[S05] The Difficulties of Using Design Patterns among Novices: An Exploratory Study [16]
[S06] Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing [7]
[S07] The Factory Pattern in API Design: A Usability Evaluation [9]
[S08] Impact of the Visitor Pattern on Program Comprehension and Maintenance [17]
[S09] Toward Effective Deployment of Design Patterns for Software Extension: A Case Study [19]
[S10] Do Maintainers Utilize Deployed Design Patterns Effectively? [20]
[S11] Documenting pattern use in Java programs [28]
[S12] A controlled experiment comparing the maintainability of programs designed with and without Design Patterns: a replication in a real programming environment [30]
[S13] Do Design Patterns Improve Communication? An Experiment with Pair Design [29]
[S14] Work Experience versus Refactoring to Design Patterns: A Controlled Experiment [21]
[S15] The Use of Architectural Patterns in the Agile Software Development of Mobile Applications [15]
[S16] Patterns in Learning to Program - An Experiment? [22]
[S17] The Value of a Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment [14]

The 11 papers includes in the mapping study appear as S03-S13 [31]. Two replication studies are included: S11 replicates S03; and S12 replicates S04. Four additional replications of S04 were recently reported in a 'Joint Replication Project' [23].

Paper S01 presented the results of three different patterns-related experiments. In this paper, we refer to these experiments as S01-A, S01-B, and S01-C. In total, we discuss 19 distinct empirical studies of software patterns represented by the 17 papers. Five studies are case studies (S01-A, S02, S05, S09 and S15) and the remaining 14 studies are controlled experiments. This set contributes insight into experimental and case study design and presents an opportunity to synthesize seemingly conflicting results to address questions related to design pattern applicability and usability. However, this number is too few to support a meta-analysis across this domain.

4. RESULTS

We now summarize the results of our survey. In Section 5, we discuss and interpret these results.

4.1 Participants Demographics

Research question RQ-1 asks, what participant demographics vary in the sample population across these studies? Table 1 summarizes the demographic categories that we collected across each study.

Identifying suitable participants is an important component of an empirical study design. Eighteen of the empirical studies in our survey involved undergraduate and graduate students enrolled in a computer science degree program. The study S12 (a replication of study S04) was conducted in a real programming environment with professional programmers. Study S12 was the only study to offer monetary incentives to participants. These incentives varied with the participant's experience level as rated by their employers. Study S01-B involved a mix of undergraduate students and professionals. The remaining studies either employed volunteers (S16) or coursework credit, such as an assignment grade or extra credit. Participants could leave the study at any time without a penalty. Coursework credits were only furnished at completion.

In some studies, participants were asked to assess their expertise levels, including familiarity with design patterns (discussed in Section 4.3). The self-assessments were measured by time period (academic or work experience), lines of code (LOC) written, and self-perceived personal rating in comparison to other students or programmers. These assessments were based on responses to questionnaires provided to the participants. In study S16, investigators used the students' performance in a prior programming course to establish a performance baseline. In study S12, experience level of participants was rated by their employers.

Table 1. Details Related to Participants

Participant Sample Size	• < 20 (S01-A, S01-B, S05, S07, S13, S15, S16, S17) • 20 - 50 (S01-C, S02, S04, S06, S08, S11, S12) • 51 - 100 (S03, S09,) • >100 (S10, S14)
Participant Role	• Undergraduate (S01-A, S03, S05, S10, S11, S14, S16) Graduate (S02, S03, S06, S08, S09, S13, S14, S15, S17) • Professional (S01-B, S04, S06, S12)

Experience Level	• Expert • Medium • Novice
Incentives	• Volunteer (S16, S01-A) • Coursework Credits • Monetary (S12)
Grouping	• Based on experience category • Based on task assigned • Based on the preparatory material given

In Table 1, most studies consisted of 50 participants or less and were conducted using coursework credit. Experience level of participants was considered relative to the other participants in the study.

4.2 Pattern Presentation to Participants

Research question RQ-2 asks, how are the patterns documented and presented to the participants? In our survey, investigators provide participants with preparatory material to orient the participants to the task. Table 2 sumarizes the preparatory materials used in the included studies. We further distinguished the following methods to deliver material to participants:

- *Shared Introduction*. All participants receive the same prepatory material at the start of the study. Materials were delivered as oral or written introduction, short course, or applied pattern examples. This approach is used to measure performance differences due to different experience levels (experts vs. novice).
- *Shared Stimulus*. All participants receive the same material during the study execution, but after performing one or more tasks. Materials were delivered as a short or long course. This approach is used to measure the impact of training on participant performance.
- *Varied Stimulus.* Two groups of participants, wherein one group receives more detailed prepatory material than the other group. This approach is used to measure differences in pattern adoption and applicability based on the material provided.
- *Full Factorial*. Two groups of participants, wherein one group receives pattern-based material (treatment) and the other group receives non-pattern-specific material (control), such as code refactoring goal. The information content of the non-pattern-specific material may be the same as the pattern-specific material but presented in general terms. In addition, participants are sub-divided using all combinations of material and participant expertise level (i.e., a full factorial experimental design).

Table 2. Type of Preparatory Material

Presentation Technique	Study Reference Number
Tool	S01-A, S01-B
Pattern Library	S01-A, S06
Video Tutorial	S01-A
Oral Introduction	S01-C, S09, S10, S13
Written Material (during the conduct of study)	S05, S06, S13, S15, S16, S17
Short Course	S03, S04, S05, S09, S10, S11, S12, S14, S15, S16
Long Course	S02, S13
Instantiated Patterns	S01-B, S03, S04, S08, S09, S10, S11, S12, S13
Documented Patterns	S03 (code), S10 (design), S11(code), S14 (code, prior to refactoring)
Multiple Examples (in varying context)	S07, S15
Real World Example	S07, S15, S17
Expert Guidance	S15

In some studies, participants were provided a tool to help with pattern selection and application. While most studies employ a short course to orient participants with the patterns, two studies employ a long course in which participants solved a set of problems before and after the course (a share stimulus study). Studies of software maintenance provided instantiated patterns, which consist of one or more design patterns applied to the given design or code. In study S08, the participants were not informed about the purpose of the study (to evaluate design pattern application). Two studies (one replication of the other) utilized the concept of pattern-specific documentation, such as code comments or taglets, to highlight the role of different classes and interfaces in the instantiated pattern. One experiment documented the patterns in the UML design.

4.3 Problem Presentation to Participants

Research question RQ-3 asks, how were the problem descriptions presented to the participants? For each study, we identified task types for the problem provided to participants as either: a *selection task*, in which participants must select a relevant pattern to solve the problem and apply the pattern; an *application task*, in which participants are provided the pattern and evaluated for their ability to apply the pattern; or a *maintenance task*, in which participants are asked to modify existing code with instantiated patterns. Pattern comprehension is an important part of performing a maintenance task involving patterns. Understanding how patterns are instantiated in an existing system can help in identifying how to modify the system. Moreover, some studies emphasize design-related software lifecycle tasks while others emphasize implementation. Table 3 summarizes our analysis for task type and lifecycle emphasis.

Table 3. Task Categorization

Category	Implementation	Design
Selection Task	S01-C, S02, S15, S16	S01-A, S01-B, S02, S05, S06
Application Task	S01-C, S02, S07, S09, S15, S16	S01-A, S01-B, S02, S05, S06, S17
Maintenance Task	S03, S04, S10, S11, S12, S13, S14	S08

In addition to task category and lifecycle emphasis, problem descriptions had other differentiating factors that we present in Table 4.

Table 4. Problem Presentation Details

Number of Tasks	• 1 (S01-B, S01-C, S09, S17) • 2 (S02, S03, S06, S08, S11, S13, S15) • 3 (S01-A, S05, S10, S14) • 5 (S07) • 6 (S16) • 9 (S04, S12)
Task Size	• KLOC
Task	• Same to all (S01-A, S01-B, S03, S05, S06,

Differences	S08, S09, S11, S13, S16, S17) • Selected by participants (S01-C, S02, • Pattern vs. non-pattern version (S04, S07, S08, S12, S14) • Different problem sets (S10, S14, S15)
Information Differences	• Same to all (S05, S09, S10) • Pre- vs. post training (S02, S04, S12, S13) • Pattern vs. non-pattern information (S01-B, S03, S06, S07, S11, S16, S17) • Less vs. more information (S15, S17)
Problem Domain	• Object-oriented (S02, S03, S04, S05, S07, S08, S09, S10, S11, S12, S14, S16) • Software architecture (S15, S17) • Version control (S13) • Ubiquitous computing (S06) • Collaborative learning (S01-A, S01-C) • Simulation environment (S01-B)
Technical Constraints	• Use of special tool or framework (S01-A, S01-B, S01-C) • Use of specific language
Ending Criteria	• Time-based (S01, S02, S03, S06, S09, S14, S15, S16) • Completion-based (S04, S05, S07, S08, S17)

Twelve studies were specific to object-oriented design patterns. Two studies were related to software architecture. Other problem domains are also listed in the above table. Most of the studies provided a time-limit for the participants to complete the task. However five studies provided no time-limit and allowed participants time to complete the task.

4.4 Evaluation Criteria after Pattern Application

Research question RQ-4 and RQ-5 ask, what are the evaluation criteria used in the experiments and what measures are used to assess these criteria in the observable outcome? Table 5 presents the complete overview of the 10 different evaluation criteria and the 23 different measures. These can be thought of as the dependent variables in an empirical study design. Example usage illustrates ways in which these criteria have been used in the empirical studies.

Some of the evaluation criteria may be mapped to quality attributes such as reusability, flexibility, understandability, functionality, extendibility and effectiveness as discussed [3] in the context of object-oriented design quality.

4.5 Reported Observations

In addition to answering the research questions in Sections 4.1 through 4.4, we summarize key observations reported in the studies below. In Section 5, we synthesize these findings to identify several implications that affect study design.

4.5.1 Studies in Pattern Selection and Application

S01-A: Novices outperformed more experienced students in terms of quality by the time they were completing the final of the three tasks. However, the experienced students took less time with each problem. Novices did not get any faster.

S01-B: The control group (with both novices and experts) couldn't finish the task in time. Novices struggled with understanding the tool while experts got lost in details. The treatment group (both novices and experts) were able to complete the design task. Novices provided efficient solutions and learned through overcoming mistakes. Experts had to overcome the additional cognitive load introduced by the conflict between their existing mental patterns and the presented design patterns.

S01-C: Use of framework introduced a steep learning curve addition to cognitive load. Groups using framework were eventually able to finish the assigned task with assistance in selecting applicable patterns. Groups using patterns without a framework were able to select applicable patterns on their own and solve the problems with relative ease, outperforming the treatment group.

S02: Code without patterns found more difficult to maintain and involved complex conditional logic and higher coupling. Even successful in applying a pattern, participants found it difficult to provide a rationale for using a particular pattern. Program size increased with the use of patterns. Polymorphism, a theme present in most of the patterns, leads to lose coupling and reduces complexity as observed by the participants.

S05: Two main types of errors were identified indicating the difficulties faced by novices in terms of pattern selection and application.

Pattern-Selection Errors: Inability to identify the most suitable pattern for a given problem. One example of pattern application might not be enough to assist novices in correctly reapplying patterns in a different problem. Applicability of some patterns found to be easier to identify (e.g., Strategy) than others (e.g., Factory). This might be in part due to the wording used in the problem statement.

Pattern-Application Errors: a) Mapping from pattern to actual design found to be the major challenge for novices; b) Misrepresentation errors such as use of concrete class instead of abstract class or interface, difficulty in identifying suitable operations or translating from generic operations in pattern to specific operations in the problem; c) Incomplete design in terms of missing key classes or operations.

S06: Experts applied patterns more effectively than novices. Patterns found to help novice designers in coming up with design idea and in explaining their ideas to others. For experienced designers unfamiliar with the domain, patterns helped in understanding the domain and communicating ideas. Patterns also helped in avoiding some design problems. Designer, without the knowledge of patterns may need to revisit and modify the design more frequently. They might also overlook important tradeoffs documented in a pattern description. None of the group employed the privacy patterns effectively and treated privacy as a secondary issue.

S07: Using factories for object construction (Factory pattern) found more time consuming than use of constructors. Presenting factories as part of the framework not found helpful in improving efficiency of task completion.

S09: Adhering to the theme of a pattern resulted in higher design quality measured in terms of following the Open-Closed Principal (OCP). Benefits in design quality not observed in solution without patterns or violating the theme of the pattern. Identified a need to explore factors that might play a role in effective deployment of design patterns.

S15: The case given more specific examples adopted the patterns in contrast to the other case given less detailed material. Case without the use of architectural patterns had significantly higher complexity in the resultant application (measured in terms of Cyclomatic complexity). The multiple case study highlights that

Table 5. Evaluation Criteria and Observable Measures

Evaluation Criteria	Observable Measures	Example Usage	
Quality	Subject matter expert evaluation	On a scale of finite discrete values, how do experts evaluate the overall quality of the deliverable	S01-A, S01-B, S01-C, S02, S05, S06
	Adherence to design principles	How the State pattern helps in complying with OCP (Open-Closed Principal)	S09
Efficiency (in problem solving)	Time to complete	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	S01-A, S01-B, S01-C, S03, S06, S07, S12, S13, S16, S04, S11, S14, S17
	Learning efficiency (ease of understanding and performing the task)	In an experiment with tasks of progressing levels of difficulty, what is the spread of achievement among participants in a given time frame, using the knowledge design patterns as a control variable	S01-A, S01-B, S01-C, S03, S06, S07, S12, S13, S16, S08
Correctness	Useful / Working Solution	In a maintenance task involving two functionally equivalent implementations with and without pattern, how useful is one approach in comparison to the other in assisting correct maintenance	S01-B, S01-C, S04, S05, S06, S09, S10, S11, S12
	Number of errors	When performing a maintenance task, assess if the use of patterns lead to less number of errors as compared to a maintenance task without employing design patterns; Assess the difference in number of errors made by novice using patterns vs. experienced participants not using pattern	S03, S04, S06, S10, S11, S12
	Number of failing tests	In maintenance, identify functional problems in deliverable; In refactoring, ensure functional equivalence	S14
Complexity	Cognitive load (qualitatively assessed)	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	S01-A, S01-B, S01-C, S02
	Eye focus (quantitative assessment of focus of attention during the task)	Using eye tracker to determine whether participants focused on the pattern-specific code while performing maintenance task or spent more time understanding non-pattern code	S08
	Cyclomatic complexity (number of linearly-independent paths)	Comparing Cyclomatic complexity of deliverables to identify cases with complex components. Higher number correlates with higher risk of defects	S15
Completeness	Requirements fulfillment	When performing maintenance tasks, how many of the required tasks were accomplished when using patterns vs. non-pattern approaches	S01-B, S03, S06, S07, S09, S10, S11, S17
Usability	Pattern selection	Assess the use of tools with support for pattern selection in helping participants	S02, S05, S09, S10, S16, S17
	Pattern application	Assess the use of multiple pattern usage examples to improve patterns applicability	
Communication	Oral communication	How pairs communicate with and without design patterns knowledge	S06, S13, S15
	Documentation	How patterns knowledge aids in capturing design rationales	S15
Creativity	Subject matter expert evaluation	a) HCI experts assessing the creative aspects of a deliverable	S06
Modularity	Coupling	Computed based on the deliverables in a programming task	S02
	Cohesion		
	WMC (Weighted Methods per Class)		
Size	LOC (Lines of Code)	Comparing 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.	S02
	NOC (Number of Classes)		
	NOO (Number of Operations)		
	NOA (Number of Attributes)		

providing pattern-specific material should be augmented with concrete examples of the use of patterns in similar problems and specific platform. Patterns enabled the teams early on to capture the core architectural components and use them for refactoring the architecture. Use of patterns also found to be helpful in communicating the architecture description and documenting the rational and design.

S16: Two groups (control without pattern knowledge; treatment with pattern knowledge) were given tasks of progressive level of difficulty. Both groups reached the expected level of task completion. Number of participants is considered less to draw a statistically significant interpretation. Researchers also stressed the need to increase the level of participation and motivating participants towards task completion.

S17: Participants who were given the full Usability-Supporting Architecture Patterns (USAP) provided complete solutions in terms of responsibilities considered in redesigning the architecture. Group given only the task description considered one-third of the responsibilities as compared to the group with detailed information and examples. Patterns found helpful in remembering and considering all the relevant responsibilities.

4.5.2 Studies in Maintenance

S03: Participants required more time on task with the presence of additional pattern-specific comments but produced more correct solutions. Quality of the solutions found to be independent of time required to solve them. Less correct solutions with Non-pattern-specific comments in less time. Time needed to correct these solutions in a real setting will offset the gain in time. Comparing only correct solutions, patterns group required less time for maintenance task and avoided mistakes as compared to other group. Benefits may be more pronounced in larger and more complex maintenance tasks.

S04: Grouping of participants in PAT (using Pattern-based solution) and ALT(using alternate, simpler solutions). All groups given same set of problems in different orders, half before pattern course (PRE), and half after pattern course (POST). All combinations of PAT/ALT and PRE/POST involved in grouping to study the effect of each factor. Using a pattern where a simpler solution would suffice found harmful in terms of program understandability and maintenance in some cases (Observer pattern). Whereas no significant differences observed in other cases (Visitor and Composite pattern). Use of Decorator pattern found to assist in program modification, however calling the modified functionality was more error prone due to delocalization of functionality. Only minor changes in the complexity of the maintenance task observed with the use of Composite pattern as it introduced less modifications in the program structure.

S08: Familiarity with patterns and UML found helpful in comprehension and maintenance tasks.

S10: While performing maintenance tasks on programs containing design patterns, developers are likely to utilize the relevant design pattern to accomplish the task. The nature of the maintenance task also affects the utilization of a design pattern. Utilizing existing design patterns during maintenance may lead to less faulty code.

S11: In this replication study, participants were less experienced than the reference study. However, it was found that pattern documentation in code is still useful in increasing the efficiency during maintenance activities. As compared to the reference study, there was an improvement in time which may be attributed to the fact that this was a web-based exercise, as opposed to paper-based.

S12: Results of the replication study were significantly different from the original study. Harmful effects of Observer pattern reported in the original study were not found in the replication. A strong negative impact of Visitor pattern found in the replication where no such indication was made in the original study. Fewer harmful effects found in using Decorator pattern. Results related to Composite and Factory pattern were consistent with the original experiment. More intuitive patterns (e.g., Observer, Decorator) generally do little harm, in terms of maintenance, even if used unnecessarily. Awareness of good design principles is significant in addition to the knowledge of design patterns.

S13: Significant differences in terms of communication observed when tested before and after provision of training material (pretest and posttest). In pretest, the explanation phase was small or even non-existent whereas rest of the communication was dominated by one individual (expert or novice). After 3-months design patterns course, there was a clear explanation phase at the start while remainder of the working time involved balanced communication. In terms of pair programming, a common understanding of problems and solutions can lead to better collaborative behavior among the pairs.

S14: Patterns introduced in example application after refactoring include Composite, Decorator, Factory Method, and Observer (top four most frequently occurring patterns according to literature). Refactoring to incorporate design patterns can improve the efficiency of maintenance tasks in terms of time spent and number of errors and can account for the additional time required to refactor This observation holds irrespective of the level of expertise of participants.

5. DISCUSSION

In this section we discuss some observations that may assist in study design and interpretation of results.

5.1 Study Execution Challenges

Some of the problems during the conduct of a study, as discussed in the literature, are given below:

5.1.1 Participant Attrition

Participant attrition means the reduction or decrease in the number of participants during the course of a study execution. Fifteen of the 19 studies involve less than 50 participants. In study S16, initially 84 students signed up for the experiment. The number decreased to 27 by the time the trial for study began. Number of participants was decreased to 18 at the start of introductory section. The results for only 10 of the participants were reported indicating that only 10 students eventually completed the study.

Assessing the applicability of patterns is more meaningful in medium to large size tasks. These tasks demand more commitment from participants in terms of time and effort. Participants may lose interest during the process or have to leave the study due to time constraints. There is a need to further explore the role of providing appropriate incentives for generating and maintaining participants' interest through the course of the study.

5.1.2 Use of Preparatory Material

Information content and presentation of the preparatory material can have a significant effect on pattern adoption and task completeness. Studies found significant differences in the level of completeness of task when one group was provided more comprehensive material (with information presented in the form of patterns) as compared to the other groups (S01-B, S03, S11,

S17). In study S17, 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.

In general, participants were divided into pattern vs. non-pattern groups. Non-pattern group received only non-pattern-specific preparatory material. Pattern group received same pattern-specific preparatory material. In study S15, both groups were given pattern-specific material. However, one group of participants received more details about applicable patterns, augmented with multiple concrete examples in the given problem domain and platform context. The other group only received basic information about patterns without specific examples. The use of patterns was optional in the study. The first group adopted the patterns in their solution in contrast to the second group. Pattern adoption may be non-existent or even erroneous when insufficient preparatory material is provided to the participants. However, the degree of separation from the examples and the problem to be solved can affect the outcome. If the examples match the problem, then participants need only apply the patterns to match their solution to the example. More work is needed to quantify an appropriate degree of separation to distinguish success attributed to the pattern, versus success due to a well-matched example.

In study S16, both groups received the same information content and use of patterns was also not mandatory. One group was presented the information in a pattern-specific manner i.e., participants in this group were introduced to a set of patterns along with the contextual relationships between various patterns. The other group was presented the same information in a pattern-neutral manner. The study does not elaborate on how the equivalence of information content (for pattern-specific and pattern-neutral representations) was established. In both groups, participants did not sufficiently utilize the preparatory material provided. In general, participants in the group with pattern-specific information did not utilize the patterns in their solutions. There is a need to further investigate how to steer the participants towards using material provided during the study without influencing the study outcomes.

5.1.3 Understanding Task Rationale

Task rationale or task goal affects pattern selection and adoption. Participants who did not understand or did not accept the task rationale or goal had difficulty in adopting the pattern. In study S02, participants who were provided the patterns were successful in applying the patterns, however, they had difficulty providing the rationale for using the patterns. In pattern selection tasks, however, problem solvers must have some knowledge of the goal to select an appropriate pattern (or schema) [27]. Thus, pattern application tasks side-step this issue, as evidenced in study S02. In addition, participants in studies S04 and S07 did not accept or realize the goal (improve maintainability) and instead found the patterns counterintuitive, because the extra work required to apply the pattern exceeded their motivation to achieve the goal.

It is important to design tasks that can realistically measure the applicability of patterns in situations that align with the objectives of the pattern.

5.1.4 Time on Task

Time to complete a task is a confounded measure of efficiency. Several studies used time to completion as an evaluation criterion (see Table 5), yet two studies suggest this criterion yields mixed results. In study S01-A, experts perform faster, however, novices outperformed experts despite taking longer to learn and apply the patterns. In study S03, participants who received more pattern-specific guidance required more time to perform the task but yielded more correct solutions.

Time can serve as a suitable measure of efficiency if: a) only correct solutions are compared; b) time needed to fix the errors in incorrect solutions is accounted for while comparing them with correct solutions. Assessing the construct validity, as considered in study S08, can help in identifying suitable observable measures for a given evaluation criteria.

5.1.5 Technical Considerations

In some of the experiments involving specific frameworks, participants faced technical issues e.g., unfamiliar error messages, lack of documentation to understand the errors, lack of training to resolve the errors quickly. In study S01-B, for example, time spent in resolving such problems due to unfamiliarity with the framework, hindered completion of the task. While one might try to improve the training to address this problem, reducing the need to use complex tools or frameworks to complete the study removes interference and any confounds attributable to these technical distractions.

5.2 Improving Pattern Use

The aim of these experiments is to improve pattern use: whether it be by identifying demographic traits of individuals likely to correspond to improved performance, or to design effective training materials. We now discuss this aim in three separate regards: pattern selection; pattern instantiation; pattern comprehension and maintenance.

5.2.1 Pattern Selection

Pattern selection means identifying suitable patterns that are applicable to a given problem description. This task was especially difficult for participants. In particular, a single example application may not be sufficient to enable the novices to identify the pattern in a different problem setting (S02, S05). Certain patterns are easier to identify when applicable (e.g., Strategy) than other patterns (e.g., Factory). Whereas the strategy pattern evokes the notion of replaceable code during runtime, similar to common plugin-based architecture, the factory pattern is more nuanced as means to instantiate classes during runtime through method invocation. The difference in participant abilitiy may relate to the presence or absence an existing mental model. Wording in a problem statement at times provides clues regarding applicable patterns (S05). This wording may unavoidably bias participants to select the appropriate pattern. Finally, we found that providing multiple, domain specific examples of pattern instantiation helps in pattern selection (S07, S15).

In some of the experiments, participants were given a light-weight tool with built-in support for pattern selection (S01-A). In those settings, novices were able to perform better with patterns and even outperformed experienced users in some cases in terms of design quality. This suggests that tool support for pattern selection can help novices in selecting the most appropriate pattern for the problem at hand.

5.2.2 Pattern Application

Pattern application is the process of mapping the pattern description or template to a concrete pattern instance in design or code. This step often requires customization of the pattern in accordance with the context of the problem. Participants might need to instantiate patterns once they have selected an appropriate pattern. They may also be given a pre-selected pattern and required to apply it to a given problem.

In many cases, this was found to be a major challenge for novices as they struggled while mapping from pattern to actual design. In the context of object-oriented patterns, one experiment (S05) identified that the mapping process led to misrepresentation errors such as incorrect use of abstract classes, concrete classes and interfaces in design. Participants found it difficult to identify appropriate operations or to translate from generic operations given in the pattern to specific operations applicable to the problem. Familiarity of participants with design languages, such as UML, was found useful when comprehending patterns (S08). This may suggest a basic understanding of object-orientation can impact results.

Another problem related to pattern instantiation was incomplete designs where key classes and operations were missing. This problem further suggests that basic design knowledge and understanding of classes and relationships is important for successful instantiation of patterns, specifically object-oriented patterns.

5.2.3 Pattern Comprehension and Maintenance

Pattern comprehension and maintenance is the process of understanding and modifying existing designs with already instantiated patterns. Comprehension and maintenance activities may be viewed as two separate tasks, however, comprehension of the existing pattern in a design or code is necessary to successfully conduct a maintenance activity.

Documentation of existing patterns in design or code was found to be helpful in performing a maintenance task in terms of efficiency and correctness. Two studies evaluated the use of documenting pattern-specific comments (S03, S11), in addition, to other comments. Participants who were provided code with pattern-specific comments found it helpful to understand the task and successfully perform the maintenance activities. One study (S03) involving experienced participants performed a comparison of correct solutions. Groups with pattern-specific comments required less time for the maintenance task and were able to avoid mistakes when compared to the other groups. A replication of this study (S11) using pattern taglets (an extension of javadoc to enable pattern-specific comments) found similar results. Even though the participants were less experienced than the reference study, pattern documentation in code was still found to be useful in increasing efficiency during maintenance activities in terms of time and correctness.

Another study documented patterns in design using UML (S10). Although there were no pattern specific comments used, this presentation assisted the participants during the maintenance activity. Participants were more likely to utilize relevant design patterns to accomplish the maintenance activity. Utilization of design patterns during maintenance also produced less faulty code. The nature of a maintenance task may also affect whether the participants will utilize existing design patterns or not.

Patterns are found useful for maintenance activities. When participants are aware of the existence of patterns in a system, they are more likely to utilize these patterns during maintenance. Pattern documentation (in the form of design or comments in code) makes it easier to locate relevant classes and their collaborations in an instantiated pattern.

5.3 Understanding Differences between Experts and Novices

Some of the studies that were included in our literature survey highlighted the differences between novices and experts while working with design patterns. It might be expected, as observed in some studies, that people with more knowledge of design patterns will be able to utilized the patterns with facility as compared to those with less knowledge of patterns. However, in some studies, experienced designers not familiar with a set of software patterns found it harder than novices to successfully utilize these patterns (S01-A, S01-B). Following subsections discuss these observations in the context of the study design and the observations recorded.

5.3.1 Internalization of Knowledge

Increased cognitive load can lead to decrease in performance. Sweller describes the importance of keeping problem solving tasks as simple as possible to avoid over burdening cognitive load [27]. We observed in a few study designs situations where the participants cognitive load (the number of items they had to remember) had negative effects on performance. In study S01-B, experts who approach problem solving with their own mental model had to overcome their preconceptions when using design patterns to solve the problem in a new way. In addition, this study employed use of a framework that decreased performance: novices dealt with a learning curve, whereas experts were distracted by the many unrelated details made available in the framework. In study S01-C, the participants were required to use a framework that had a steep learning curve, which distracted from the pattern task.

Novices may be able to internalize the knowledge captured in a design pattern more effectively if the cognitive load is minimized while presenting and documenting patterns.

5.3.2 Unlearning and Re-learning

Discuss, in the context of empirical evaluation, why experts not familiar with patterns find it harder at times to work with patterns. They have their own mental schemas of knowledge and the new pattern may present a different schema than their own. Experts would then need to unlearn or adapt their own schemas to understand and apply the patterns which introduces additional cognitive load. Whereas novices may not have such existing schemas and would experience cognitive load to build new schemas instead of to unlearn or adapt their existing schemas.

In study S01-A, aimed at assessing the cognitive learning efficiency through use of design patterns, novices outperformed the more experienced students in terms of quality as they completed a set of three design problems. The experienced students took less time with each problem whereas such as reduction in time to solve a problem was note observed for novices. However, in study S01-B, with control (consisting of novices and experienced designers without patterns) and treatment (consisting of novices and experienced designers with patterns) groups, both novices and experiences designers using patterns performed better than the control group.

Experts had to overcome the additional cognitive load introduced by the conflict between their existing mental patterns and the presented design patterns. Similar findings were reported in relation to creativity of participants in the domain of decision support systems [6].

6. CONCLUSION

We have conducted a survey of literature on empirical evaluation of software design patterns. The survey involves 19 distinct studies reported in 17 papers. From these studies, we have extracted information that can be helpful when designing an empirical study for assessing efficiency and effectiveness of software patterns. We have identified 17 independent variables in three different categories (5 related to participants demographics; 3 related to pattern presentation; 9 related to problem presentation). We also extracted 10 different evaluation criteria with 23 associated observable measures. We synthesized the reported observations to identify challenges in study execution as well as discussion on improving pattern use. Our literature survey can be used by software design patterns researchers and developers as an experimental design roadmap when evaluating patterns for efficiency and effectiveness.

7. ACKNOWLEDGMENTS

We thank the NCSU Realsearch Group for their help with this paper. A special thanks to Ben Smith for his help with voting on the papers.

8. REFERENCES

[1] B. Adelson and E. Soloway, "The role of domain experience in software design," *IEEE Transactions on Software Engineering*, vol. 11, no. 11, pp. 1351 - 1360, Nov. 1985

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language*. New York: Oxford University Press, 1977.

[3] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment.," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, 2002.

[4] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using mapping studies in software engineering," in *20th Annual Psychology of Programming Interest Group Conference (PPIG 2008)*, Lancaster University, UK, 2008.

[5] A. Chatzigeorgiou, N. Tsantalis, and I. Deligiannis, "An empirical study on students' ability to comprehend design patterns," *Computers and Education*, vol. 51, no. 3, pp. 1007-1016, 2008.

[6] P. K. Cheung, P. Y. K. Chau, and A. K. K. Au, "Does knowledge reuse make a creative person more creative? ," *Decision Support Systems*, no. pp. 219-227, 2008.

[7] 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.

[8] P. Clements and L. Northrup, *Software Product Lines: Practices and Patterns*. Reading, PA: Addison-Wesley Professional, 2001.

[9] 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.

[10] M. Fowler, *Analysis Patterns: Reusable Object Models*. Menlo Park, CA: Addison Wesley Longman, Inc, 1997.

[11] M. Fowler, *Analysis Patterns: Reusable object models*. Reading, MA: Addison-Wesley Professional, 1997.

[12] M. Fowler, *Patterns of Enterprise Application Architecture*. Reading, MA: Addison-Wesley Professional, 2002.

[13] E. H. Gamma, Richard; Johnson, Ralph; and Vlissides, John, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1995.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] G. Kolfschoten, S. Lukosch, A. Verbraeck, and Edwin Valentin, "Cognitive learning efficiency through the use of design patterns in teaching," *Computers and Education*, vol. 54, no. 3, pp. 652-660, April 2010.

[19] 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.

[20] 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.

[21] T. H. Ng, Cheung, S.C., Chan, W.K., and Yu, Y.T. 2006. . In Proceedings of the "Work Experience versus 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.

[22] R. Porter and P. Calder, "Patterns in Learning to Program - An Experiment?," in *Sixth Australasian Conference on Computing Education (ACE'04)*, 2004.

[23] L. Prechelt and M. Liesenberg, "Design Patterns in Software Maintenance: An Experiment Replication," in *Second International Workshop on Replication in Empirical Software Engineering Research (RESER'11)* Freie Universität Berlin, 2011.

[24] 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, no. 12, pp. 1134-1144, 2001.

[25] L. Prechelt, Unger-Lamprecht, P. B., M., and W. F. Tichy, "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance," *IEEE Transactions on Software Engineering*, vol. 28, no. 6, 2002.

[26] D. C. Schmidt, M. Fayad, and R. E. Johnson, "Software patterns," *Communications of the ACM,* vol. 39, no. 10, Oct. 1996.

[27] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive Science,* vol. 12, no. pp. 257-285, 1988.

[28] M. I. P. o. Torchiano, "Documenting pattern use in java programs," in *International Conference on Software Maintenance (ICSM'02)*, Montreal, Canada, 2002, pp. 230-233.

[29] B. Unger and W. Tichy, "Do design patterns improve communication? an experiment with pair design," in *I nternational Workshop on Empirical Studies of Software Maintenance*, 2000, pp. 1-5.

[30] 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, no. 3, pp. 149-195, Sept 2004.

[31] C. Zhang and B. Budgen, "What do we know about the effectiveness of Software Design Patterns?," *IEEE Transactions on Software Engineering,* no. July 2011.