What Do We Know about the Effectiveness of Software Design Patterns?

Cheng Zhang and David Budgen, *Member*, *IEEE Computer Society*

Abstract—Context. Although research in software engineering largely seeks to improve the practices and products of software development, many practices are based upon codification of expert knowledge, often with little or no underpinning from objective empirical evidence. Software design patterns seek to codify expert knowledge to share experience about successful design structures. **Objectives.** To investigate how extensively the use of software design patterns has been subjected to empirical study and what evidence is available about how and when their use can provide an effective mechanism for knowledge transfer about design. **Method.** We conducted a systematic literature review in the form of a mapping study, searching the literature up to the end of 2009 to identify relevant primary studies about the use of the 23 patterns catalogued in the widely referenced book by the "Gang of Four." These studies were then categorized according to the forms of study employed, the patterns that were studied, as well as the context within which the study took place. **Results.** Our searches identified 611 candidate papers. Applying our inclusion/exclusion criteria resulted in a final set of 10 papers that described 11 instances of "formal" experimental studies of object-oriented design patterns. We augmented our analysis by including seven "experience" reports that described application of patterns using less rigorous observational forms. We report and review the profiles of the empirical evidence for those patterns for which multiple studies exist. **Conclusions.** We could not identify firm support for any of the claims made for patterns in general, although there was some support for the usefulness of patterns in providing a framework for maintenance, and some qualitative indication that they do not help novices learn about design. For future studies we recommend that researchers use case studies that focus upon some key patterns, and seek to identify the impact that their use can have upon maintenance.

Index Terms—Design patterns, systematic literature review, empirical software engineering

1 Introduction

Although software engineering is intrinsically an "empirical" subject, this aspect has so far largely manifested itself through the (relatively informal) reuse of "experience" and expert opinion rather than through the use of the outcomes from systematic empirical investigations [37]. Various authors have noted this: Back in 1997, Whitley commented that "a common pattern in software engineering research is the development of system-building techniques, such as object-oriented design, which are strongly advocated in the absence of evidence" [61], while, stemming from their systematic surveys of the literature, Glass et al. have made similar observations about the emphasis upon advocacy and the predominance of analysis as the means for evaluation [23], [24].

The use of software design patterns for designing object-oriented (OO) systems would appear to fit into this category. The large number of papers and books published about patterns supports the view that the concept of the design pattern is valued by many experienced developers as a categorization of recurring issues (and solutions), as well as providing a widely used vocabulary for discussing design. However, the available literature on patterns appears to be largely in the form of "advocacy" or "experience," and much of it is also focused upon identifying and documenting patterns rather than reporting and analyzing experience with *using* them. Probably the most widely known textbook is that by the "Gang of Four" [21] (usually abbreviated to *GoF*, as we will do here)—and while this employs a template for cataloguing patterns that has been widely adopted, the template offers little that would encourage the notion of providing evidence about where and when a given pattern might best be used, beyond the headings "Applicability" and "Known Uses."

Like most software engineering practices, design patterns have been distilled from the experiences of software developers. For patterns to provide an effective vehicle for conveying design experience, it is necessary for them to be usable by less experienced designers and also for that use to lead to successful designs. In this paper, we describe a *mapping study* we have undertaken to determine the scale and extent to which empirical studies have been undertaken to determine how effective patterns are as a knowledge transfer mechanism and the forms of *evidence* for this.

The *evidence-based* paradigm is concerned with systematically and objectively finding and aggregating all of the available empirical data on a chosen topic, on the basis that the outcomes from such a "secondary" study can reduce any bias that might occur in the outcomes from individual "primary" studies. The core tool of the evidence-based paradigm is the *Systematic Literature Review* (SLR), often abbreviated to "Systematic Review" [38], [48], which provides a framework for systematically searching the literature, extracting the data, and performing the necessary analysis.

• *The authors are with the Science Laboratories, School of Engineering and Computing Sciences, Durham University, South Road, Durham DH1 3LE, United Kingdom. E-mail: {cheng.zhang2, david.budgen}@durham.ac.uk.*

Manuscript received 1 Apr. 2010; revised 3 June 2011; accepted 14 July 2011; published online 29 July 2011.
Recommended for acceptance by M.-A. Storey.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-04-0094.
Digital Object Identifier no. 10.1109/TSE.2011.79.

The evidence-based paradigm has become strongly established in clinical medicine, where the use of secondary studies has become well codified and widely used, with conduct and publication of studies coordinated through the "not for profit" *Cochrane Collaboration.*[1] Between 2004, when its adoption for use in software engineering was proposed by Kitchenham et al. [35], [18], and mid-2008, over 50 relevant published SLRs and mapping studies have been identified [36], [39].

However, there are some differences between the domains of clinical medicine and software engineering that constrain its effectiveness when employed in the latter [11]. In particular, we should note that:

- The interventions used in primary studies in clinical medicine are performed *upon* the participants, whereas for software engineering, they are performed *by* the participants. Any differences in the skills of the individual participants and the influence of the context (the specific task being performed) will increase the complexity.
- The human body, and the terminology associated with it, does not change significantly, making it easier to identify relevant keywords to employ when searching for primary studies of relevance as well as to perform longitudinal studies over a period of time. In contrast, software engineering practices are often only imprecisely defined, and may change with increasing knowledge and experience. Likewise, the terminology also may change or evolve, creating problems for the searching phase.

Both of these are significant factors when looking at studies of the design process. The first creates a particular problem for the experimenter, in that the individual skill levels of participants forms a significant confounding factor. However, the second factor is at least in part offset for the case of design patterns by the way that the predominance of the book by the *GoF* has created a relatively stable codification standard and vocabulary—although, as we observe later, some of the earlier papers did use different terminology.

A mapping study (also termed a *scoping review*) is sometimes used before undertaking a systematic literature review, in order to identify the extent of, and categorize the form of, the literature on a particular topic. Here, we have performed a mapping study to help identify those primary studies that evaluate aspects of design patterns in any way and hence to determine what forms and issues have been studied, as well as by what means. Table 1 highlights some key distinctions between a mapping study and a systematic literature review in terms of some of the main elements [40]. (The boundaries are of course not rigid, and in this mapping study we do undertake a modest amount of quality evaluation.)

In undertaking this study, we set out to address the following research questions:

- Which of the *GoF* patterns have been evaluated empirically?
- What lessons about their use, and any consequences of this, particularly regarding constraining effects upon any subsequent maintenance tasks, are available from these empirical studies?
- What further research, using which forms, might be needed to address any "gaps" in the available evidence?

1. See www.cochrane.org.

TABLE 1
Differences between Mapping Studies and SLRs

Element	Mapping Study	SLR
Goals	Classification and thematic analysis of literature on an SE topic	Identifying best practices with respect to specific procedures, technologies, methods or tools by aggregating information from comparative studies
Research question	General—related to *research trends*. Which researchers; how much activity; what type of studies etc.	Specific—related to *outcomes* of empirical studies. Of the form: "does technology/method A have property X?"
Search process	Defined by the topic area.	Defined by the research question.
Required search outcomes	Less stringent if only research trends are of interest	Extremely stringent—all relevant studies need to be identified.
Quality evaluation	Not essential	Important to ensure that the results are based on best quality evidence.
Results	Set of papers related to a topic area, categories for these, and counts of papers in each category.	Answer to specific research question, possibly with qualifiers (e.g. that results apply to novices only).

Conducting empirical studies in software engineering is nontrivial, and especially so for studies that address complex cognitive processes such as design. Hence, we did not anticipate finding other than a relatively modest corpus of primary studies that addressed issues relating to design patterns. As we explain in Section 3, despite a large number of "hits" when searching, the final set of experimental studies used for our analysis only contained a small number of papers. To assist with analysis and interpretation, we therefore widened our interpretation of "empirical" and looked to see what supplementary forms of evidence might be available. We were then able to identify a small set of "experience" reports that provided reasonably well-documented observations about specific patterns and could be used to augment our analysis.

In the rest of this paper we first discuss some of the characteristics of software design patterns, and then describe how we organized our mapping study, the outcomes from this, and the degree to which it answers the questions. Finally, we discuss the implications that our findings have for design pattern use and adoption.

2 Knowledge Transfer through Design Patterns

This section examines the issue of knowledge transfer for software design.

2.1 Knowledge Schema

Software design is widely recognized as being a "wicked" or "ill-structured" problem, characterized by ambiguous

specifications, no true/false solutions (only ones that are "better" or "worse" from a particular perspective), the lack of any "stopping rule" to determine when a solution has been reached, and no ultimate test of whether a solution meets the requirements [25], [54]. The processes for seeking solutions to such problems (in our case, software designs) cannot therefore be a procedural or "defined" process, and so must employ more empirical or "opportunistic" strategies [28], [62].

Studies of expert software designers support this view. A study of experienced and inexperienced designers by Adelson and Soloway noted a range of techniques being used, with these being dependent upon both the expertise of the designer, and also their familiarity with the problem domain [2]. In particular, they observed the use of "labels for plans" by experts, whereby a designer recognized that for part of their task they could reuse earlier design experience, and made a note of this at a relatively abstract level.

In a later study of expert designers by Guindon [25], the author discusses the observed use of a range of forms of *knowledge schema*, from the simple rule to the part solution. Such a schema incorporates a pattern that specifies both the characteristics of the problem and also the form of the corresponding solution, and hence includes the concept of labels for plans.

For object-oriented development, Détienne identifies three forms of relevant knowledge schema: application domain schemas, function schemas, and procedure schemas. Such a schema is a "knowledge structure" that represents "generic concepts stored in memory," rather than just a solution plan [17].

One of the distinctions between expert and novice designers is that the former possess a much richer and fuller set of knowledge schemas. Over the years, various techniques such as design methods and design patterns have been employed, both to help less experienced designers to develop their own knowledge schemas by acquiring additional knowledge drawn from the experiences of experts (at least partially), and also to augment the set of schemas possessed by other experts. We might also note that some authors view the second of these as being more practical and appropriate for the case of design patterns [56].

2.2 Design Patterns

The concept of the design pattern was formulated by the architect Christopher Alexander, who describes it as follows:

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.* [3]

As such, the concept is clearly one that is applicable to the design process in any domain, including software. Such behavior also occurs at different levels of abstraction, for example: Buschmann et al. have discussed the idea of "architectural patterns" that describe overall system organization [13], while the software design patterns identified in the *GoF* essentially address "part-system" issues.

Software design patterns may therefore be regarded as a means of codifying expert knowledge schemas derived from software design practice. However, there are two important assumptions implicit in such a view:

1. That knowledge schema can be codified in such a way that they can be shared between different designers. (The observational studies discussed in the previous section only considered the situation where a designer reused their own previous experiences.)
2. That reuse will lead to "better" designs. There is an obvious question here of what constitutes "better," but for the studies that we have identified there seems to be a general consensus that the key measure for this concerns the maintainability of a design.

Curiously, this first assumption does not seem to have been explored until quite recently [41]. The authors of this paper observe that knowledge schema are an internal representation of knowledge which cannot be directly shared, and that a design pattern is one means of externalizing a schema for transfer to others, and also of coming to a consensus about its nature. However, because each individual who learns a pattern must gain an understanding of the pattern to create and store their own form of schema within their own knowledge store, multiple examples of that pattern's application may well be needed to assist with this process—potentially adding a further confounding factor for experimental studies of their use. The second assumption has been noted by some researchers [8], [51] and Bieman particularly observes that: "there is very little empirical evidence of the claimed benefits of design patterns and other design practices when applied to real development projects."

For this study, the original idea was that we would seek studies that investigated the use of design patterns and see how extensively these met the claims for design patterns. In practice, it proved difficult to identify a coherent set of claims since many are entwined with a definition of what a pattern is and the comparisons implied do not readily relate to any specific measures. For example: "design patterns make it easier to reuse successful designs" is taken from the *GoF* and is really a definition of the role of a pattern with no clear sense of what measures should be related to "easier." However, one of the papers we analyzed [52] has identified the following advantages as being claimed for design patterns:

1. Using patterns improves programmer productivity and program quality.
2. Novices can increase their design skills significantly by studying and applying patterns.
3. Patterns encourage best practices, even for experienced designers.
4. Design patterns improve communication, both among developers and from developers to maintainers.

While these still lack baseline values for comparison purposes, we have adapted these as the relevant issues to explore in our mapping study, while also recognizing that they should really be separately assessed for each pattern.

To address these issues, we have focused our mapping study on identifying which patterns have been investigated empirically, the forms this has taken, the extent to which different studies agree or disagree, and how far they

address each issue. As observed above, in doing so, we have added an element of quality assessment to the analysis of the papers in the clusters in order to assess how much weight to ascribe to specific studies.

3 Finding the Evidence: The Mapping Study

A mapping study is: "designed to provide a wide overview of a research area, to establish if research evidence exists on a topic, and provide an indication of the quality of the evidence" [38].

So, while the planning phase is similar to that of a systematic review, much of the focus of a mapping study is upon the activities undertaken for the first three stages of the second phase of a review, namely:

- identification of research evidence (searching),
- selection of primary studies (inclusion/exclusion),
- assessment of study quality (bias/validity).

Any data extracted about the outcomes from the primary studies included are generally less detailed than for a systematic review, and is mainly for the purposes of classification and categorization.

For this study, since we also wanted to "profile" the patterns literature, we conducted the second (inclusion/exclusion) stage using the following three substages:

- selection of *all* papers that were about software design patterns,
- categorization of these papers in terms of their *purpose*,
- selection of papers that provided actual *empirical* data about the use of patterns.

3.1 Conduct of the Study

For the *searching* stage, the general *scope* of the study was identified as being:

- **Population:** Published scientific literature reporting software design pattern studies.
- **Intervention:** Studies involving the use of software design patterns.
- **Outcomes of relevance:** Quantity and type of evidence relating to specific instances of design patterns.
- **Experimental design:** Any form of empirical study.

Because the *GoF* [21] is regarded as the milestone textbook on design patterns, the start for the search period was chosen as 1995 (publication of *GoF*).

Searching for relevant papers was organized as a three-stage process.

- **Round 1** involved performing an electronically based search covering the period from start of 1995 until the end of 2009. After some initial prototyping of possible terms, we used the terms "software + pattern" together with the following: *experience, investigation, experiment, study, experimental, empirical, apply, use, implementation, application, investigate, investigating, experimentation, utilise, utilisation, employ, practice, survey, work, sketch, analyze, analysis, usage, exercise, implement, construct*. Where appropriate we checked that different spellings (such as "utilize" and "utilise") were recognized as equivalent by the search engines. Our choice of search terms was based upon those that inspection of other studies suggested were commonly used in the research literature, together with a dictionary check for synonyms.

  Following the advice of the guidelines for systematic reviews and in order to perform an exhaustive search [7], we selected six search engines/sources which are relevant to software engineering: ACM, IEEE Xplore, Google Scholar, CiteSeer, ScienceDirect, and Web of Science. The search queries were used with all six search engines/sources, restricting their scope to title, abstract, and keywords where the interfaces permitted this. Collectively these accessed the main digital libraries considered to be appropriate to the study.
- **Round 2** consisted of a manual search through four journals that were identified as major sources of papers (*IEEE Transactions on Software Engineering*, *Empirical Software Engineering*; *Journal of Systems & Software*, *Information & Software Technology*) covering the same period (1995-end 2009). This round was used primarily to act as a check on the reliability of the electronic searches, and in particular, upon our choice of search strings. Since the key journals were easier to identify than conferences, we confined our search to the journals.
- **Round 3** consisted of a "snowball" search, checking the references used in the empirical papers found in the first two rounds, to see if these identified any further papers.

For the searching stages of our study we employed the following *inclusion* criteria:

- papers describing software design patterns (not just empirical papers, as we wanted to be able to categorize the overall patterns literature, although only the empirical papers are actually analyzed in this paper),
- where several papers reported the same study, only the most comprehensive (usually the most recent) was included,
- where several studies were reported in the same paper, each relevant study was treated as an independent primary study;

and the following *exclusion* criteria:

- literature that was only available in the form of abstracts or *Powerpoint* presentations,
- technical reports and submitted papers.

We should note that no quality assessment was performed at this stage in order to ensure maximum coverage. Our search terms also meant that we found a wide range of papers other than simply empirical papers since a longer term concern is to determine how well the available empirical studies address the issues identified in concept papers and tutorial papers. In order to perform this analysis, we therefore needed to identify as wide a spectrum of the literature related to design patterns (both concept and specific instances) as possible.

TABLE 2
Round 1 Results by Search Engine

Search Engine	Count
ACM	118
IEEE	118
CiteSeer	47
ScienceDirect	34
Web of Science	8
Google Scholar	77
Total	402

3.2 Outcomes

The three rounds of searching identified a total of 611 papers after removing papers that did not refer to design patterns as well as any duplicates (found by more than one search engine). Table 2 provides the final counts from the initial search (Round 1) for each of the search engines. Google Scholar was used last, due to its lack of precision, and the count for this is for those papers that were not identified through the other search engines. Since we were interested in knowing the proportion of papers that had an empirical focus, the titles and abstracts were used by one of us (CZ) to classify papers (broadly). After inspection of the papers found, we adopted a scheme of classifying these by using the themes of *tutorial, support tool, empirical, construction*, and *index* to describe their focus. Our choice of these themes was based upon the categories discussed on the patterns web site (hillside.net), although some of our terms such as "construction" differ from those used there. Table 3 explains how these classifications were interpreted, together with the figures for each of these from the three rounds. We should note that most of the papers from Round 2 were from the early years, reflecting some variation in terminology during that period.

In the remainder of this section, we examine the profile for the *empirical* papers alone. These were considered to be any paper that provided some form of data about patterns and employed one of the major empirical forms most commonly used in software engineering, namely, *experiment*, *case study*, *survey*, or *experience report*.

3.3 The Empirical Papers

For our initial filtering of these papers we took a liberal interpretation of what was meant by "empirical," including papers that ranged from those that were essentially informal observational "experience" papers through to those describing controlled experiments. Our motivation here was to ensure that we did not miss any potentially relevant material.

As has been observed in other secondary studies, this process is invariably confounded by the rather casual use of technical terms in many papers. Particular examples of this are:

- Few papers that describe themselves as conducting a *case study* are actually treating this as an empirical method, such as the positivist form documented by Yin [64] that can be readily adapted for software engineering use [9], [29]. Many of them are observational narratives that could perhaps be more correctly described as "use cases."
- The term *experiment* is used very casually. While in empirical software engineering this is generally taken to refer to a randomized controlled experiment or a quasi experiment employing human participants, many authors use the term in quite cavalier fashion. This is particularly evident in papers that describe tools (such as those used to extract patterns from code) and then describe the application of the tool to a chosen software artifact as being an experiment, despite the lack of randomization or of any form of control group.

The 219 papers in the "empirical" category formed the core set for the formal systematic review process itself. Following the SLR guidelines, we performed the formal inclusion/exclusion phase as follows:

- Exclusion on the basis of title.
- Exclusion after reading the abstract.
- Exclusion after reading the full paper.

Fig. 1 provides a schematic of this process, indicating the number of papers remaining after each step. For each phase, we both performed an independent analysis of the candidate papers and then produced an agreed list. When excluding on the basis of title and abstract we sought to retain any papers that might possibly contain useful experience *about* the use of patterns, and hence took a conservative approach, retaining a paper if there was any possibility. We also calculated the *Kappa* score for interrater agreement for our initial levels of agreement. For exclusion on title this was 0.44 (moderate agreement) and for exclusion on abstracts it was 0.60 (verging closely on good). The size of the final set is also generally consistent with those found in mapping studies performed on other design-oriented topics such as [19], [27], [44], [53].

TABLE 3
Classification Themes Used and Related Counts of Papers

Theme	Interpretation	Round 1	Round 2	Round 3	Total
Tutorial	How to use patterns	107	24	24	155
Support Tool	Creating and using tools for extracting or using pattterns	62	25	22	109
Empirical	Providing some form of empirical data	143	42	34	219
Construction	How to write patterns	37	4	12	53
Index	How to index or catalogue patterns and to find them	53	3	19	75
Totals		402	98	111	611

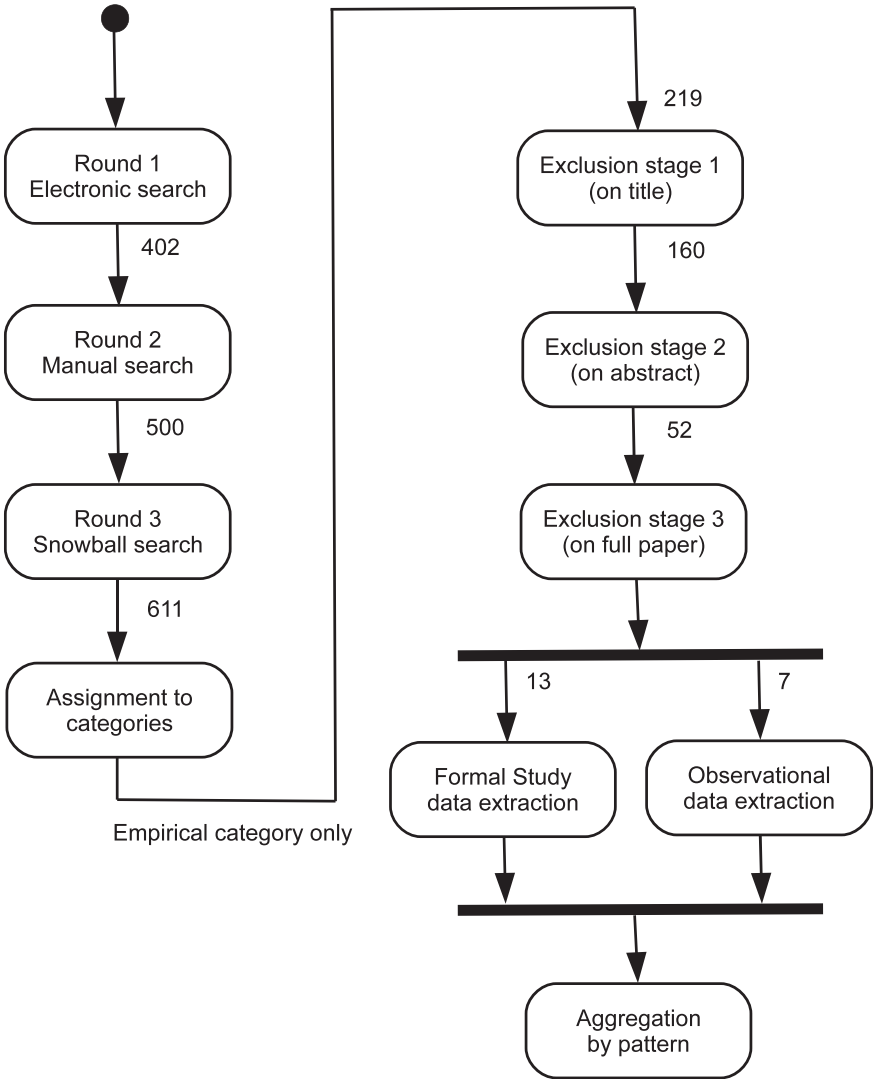


Fig. 1. Overview of the review process (figures between the boxes are the number of papers remaining at each step).

The initial filter identified nine candidate survey papers, but none of them provided suitable evidence about specific patterns. However, one survey did assess 10 quality attributes for all of the *GoF* patterns using a sample of 20 experienced developers [33].

The 13 "experiment" reports involved a range of rigor and experimental form (for this reason we will refer to these as "formal studies (FS)," coded as FS1 to FS13). These were largely published as Conference or Workshop papers, only three being published in journals.

4 The Evidence in Detail

In this section, we describe the nature of the evidence provided in the papers that described formal studies such as controlled experiments. To do so, we examine the research questions used for the studies, the way that they were organized, the specific instances of patterns that were studied, the types of participant and task, and the extent to which the outcomes of similar or replication studies agreed. One study included (FS1) did not study object-oriented patterns and so is not analyzed in this section (it was originally included because it studied the use of the pattern concept, but in another context—namely ubiquitous computing.)

During the more detailed analysis, it also became evident that FS6 [50] was a preliminary report on the first part of the study later reported in FS8, with the latter including additional data from a second group who repeated the study in a different venue. To avoid double-counting we therefore removed FS6 from the set, leaving 11 papers. There was also overlap between FS5 [49] and both FS7 and FS8, with the second experiment of FS5 being reported (more fully) in FS7, and the first in FS8 (but reporting a different count of participants). So FS5 was also removed from our analysis, although where relevant we use qualitative comments reported in FS5. FS8 (and hence FS5) also reported upon two instantiations of their experimental study, occurring in two well-separated locations, with different participant profiles, different programming languages, different working modes—and with some different results. So we have labeled these as $FS8_1$ and $FS8_2$ and, where appropriate, have treated them as being separate studies for purpose of analysis. We should also note that two of the studies (FS9 and FS11) were replications of previous studies, although with different types of participant.

TABLE 4
Research Questions Addressed

No.	Research Question(s)
FS1 [15]	How applicable is the pattern concept to ubiquitous computing?
FS2 [20]	Whether factories are detrimental to API usability when compared with using constructors?
FS3 [45]	Whether a relation exists between the use of patterns and the open-closed principle?
FS4 [46]	"Given a software system with relevant design patterns deployed and documented, how likely will its maintainer utilize the design patterns to complete an anticipated change?"
FS7 [51]	Where a solution using a pattern could be replaced by a simpler one is it still helpful to use the design pattern?
FS8 [52]	"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?"
FS9 [57]	(Replication of FS8.)
FS10 [58]	If team members have common design pattern knowledge and vocabulary, can they communicate more effectively than without these?
FS11 [59]	(Replication of FS7.)
FS12 [32]	Whether the presence of *Visitor* affects developer effort, and whether using different layouts for the UML diagram have any effect?
FS13 [31]	To identify the difficulties associated with patterns application by novices

4.1 Research Questions

We examined the papers to identify the type of research question that they were addressing, and these are listed in Table 4. A significant characteristic was that all of the experiments on object-oriented patterns were concerned with studying and modifying existing systems.

4.2 Which Patterns Have Been Studied?

All of the formal studies of OO patterns used only patterns from the *GoF* [21], and in terms of *purpose* they used a mix of the five creational, seven structural, and 11 behavioral patterns that are catalogued there, as well as including both class and object patterns (*scope*). Table 5 summarizes the frequency with which specific patterns were employed. While 15 of the *GoF* patterns were used, many were only used in a single study. The patterns that were not studied were: Builder, Prototype, Flyweight, Proxy, Interpreter, Iterator, Mediator, and Memento. Only two studies focused upon a specific pattern and its properties (FS2 and FS12). From Table 5 we can therefore conclude that only *Composite*, *Observer*, and *Visitor* have been studied very extensively.

TABLE 5
Patterns Used in the Different Studies

Pattern	Scope	Studies using	Total
Creational Patterns			
Abstract Factory	Class	FS7, FS11	2
Factory Method	Object	FS2, FS4, FS13	3
Singleton	Object	FS10	1
Structural Patterns			
Adapter	Object	FS13	1
Bridge	Object	FS10	1
Composite	Object	FS4, FS7, $FS8_1$, $FS8_2$, FS9, FS10, FS11	7
Decorator	Object	FS7, FS11, FS13	3
Facade	Object	FS7	1
Behavioural Patterns			
Chain of Responsibility	Object	FS10	1
Command	Object	FS4	1
Observer	Object	FS4, FS7, $FS8_1$, $FS8_2$, FS9, FS10, FS11	7
State	Object	FS3, FS4	2
Strategy	Object	FS13	1
Template Method	Class	$FS8_1$, $FS8_2$, FS9	3
Visitor	Object	FS7, $FS8_1$, $FS8_2$, FS9 FS10, FS11, FS12	7

4.3 Participants and Tasks

We looked at the type of participant used in each study and their degree of experience with using patterns. We also examined the way the studies were organized, the tasks assigned to participants, and the measures used. Our summaries of these are presented in Tables 6 and 7.

Details of the participants were generally reported quite fully and while many were (perhaps inevitably) taking advanced undergraduate or graduate courses, there was also quite a large contingent who had extensive programming experience and industry experience. Given the not

TABLE 6
Details of the Participants

No.	Details of Participants
FS1	16 pairs of designers, using a mix of professionals (6–8 years of experience) and graduate students
FS2	12 males, aged 18–35 with at least a year's experience of Java, 8 with professional programming experience, 6 with some experience of using the factory pattern that was the topic of the study.
FS3	98 part-time postgraduate students with an average of 5 years working in the computer industry.
FS4	215 undergraduate students taking a Java course.
FS7	29 professional software engineers from one company with average of 2.4 years of C++ experience, and with 15 having some prior knowledge about patterns.
$FS8_1$	64 graduate and 10 undergraduate students with an average of 7.5 years programming experience.
$FS8_2$	22 participants, all undergraduate students with an average of 5 years programming experience.
FS9	28 undergraduate students with limited programming experience.
FS10	15 graduate students with an average of 5.8 years programming experience.
FS11	44 professional software engineers from different companies.
FS12	24 graduate students
FS13	16 final year undergraduate students

TABLE 7
Tasks and Measures Used

No.	Study Form	Tasks Performed	Measures Used
FS1	between-subjects	Working in pairs. Task 1 to assess whether patterns help with evaluating an existing design. Task 2 to produce a design. Fixed time for both.	Task 1 used participant ratings. Task 2 used ratings from three postgraduate 'judges'.
FS2	within-subjects	5 programming tasks using factory or a constructor.	Completion time.
FS3	between-subjects	Modifying the code of a Java program.	Functional correctness and use of patterns.
FS4	between-subjects	Modifying the code of 3 Java programs.	Functional correctness and use of patterns.
FS7	within-subjects	Modifying two (from four) C++ programs.	Completion time and error count.
FS8	within-subjects	Modify design and then code for two programs.	Completion time and tasks satisfied.
FS9	within-subjects	Modify Java code. (Replicates E8.)	Completion time and tasks satisfied.
FS10	within-subjects	Expert/novice pairs performing maintenance tasks.	Use of protocol analysis to analyse communication within the pairs.
FS11	within-subjects	Modifying four programs. (Replicates E7.)	Completion time and degree of functional correctness.
FS12	between-subjects	Interpreting class diagrams.	Focus of eyes on elements of UML diagrams
FS13	within-subjects	Interpreting class diagrams.	Revised diagrams

uncommon view that the use of patterns does require a degree of maturity with object-oriented design and implementation [56], this did seem to be recognized in this set of experiments.

The way that the experimental aspects of the study were organized so as to address the research question is summarized in Table 7. Where a study is classified as being "within-subjects" in form, this was generally of the form: *pretest—training—posttest*.

One feature of Table 7 is that most of the OO studies (FS3-FS13) involved *modification* and that most of them also involved an element of *coding*. Given that many of the qualities advocated for design patterns are concerned with the more abstract activities of *design*, such a strong emphasis on coding seems surprising—although probably reflecting something of the difficulty of conducting empirical studies involving design activities. Indeed, the only studies that did not involve an element of coding were FS1, FS12, and FS13.

Finally, Table 8 summarizes the details of the different programs used for the purposes of comprehension and/or modification for those studies that used the *GoF* patterns.

TABLE 8
Programs Used in the Studies

No.	Program(s)	Language	Size
FS2	Notepad Email; Elipse Email Thingies, PIUtils, Sockets	Java	*No details provided.*
FS3	Calendar Manager (MCM)	Java	Approx. 1500 LOC
FS4	JHotDraw Calendar Manager (MCM) Hotel Management (HMS)	Java	15815 LOC/211 classes 1455 LOC/15 classes 583 LOC/10 classes
FS7	Stock Ticker (ST) Boolean Formulas (BO) Communication Channels (CO) Graphics Library (GR)	C++	343 LOC/7 classes 470 LOC/11 classes 365 LOC/6 classes 682 LOC/ 13 classes
$FS8_1$	And/Or Tree Phonebook	Java	362 LOC/7 classes 565 LOC/11 classes
$FS8_2$	And/Or Tree Phonebook	C++	498 LOC/6 classes 448 LOC/6 classes
FS9	*As* $FS8_1$	Java	*Modified, no details*
FS10	CHICO (version control front-end) TIMMIE (time/defect tracking)	Not stated	76 methods/15 classes 102 methods/13 classes
FS11	*As FS7*	C++	*Slightly different sizes.*
FS12	JHotDraw JRefactory, PADL	n/a	*No Details.*
FS13	Interactive Quiz Environment (IQE)	n/a	*Not given.*

For each study we indicate how many programs were employed and indicate their sizes (when known). Where more than one version was provided, we quote the size of the version that used patterns.

4.4 Degree of Agreement between Studies

In order to assess the extent to which the different studies find similar effects we have looked at the following two questions.

4.4.1 Replication Studies

There are two cases of replicated studies. Here our question is: "*Do the replication studies demonstrate any degree of consistency in their findings?*"

- FS9 uses the material from FS8 (a study of patterns as documentation) although extending it to use Javadoc tags, the use of which actually forms the focus of the paper. So, although this is a replication study, its purpose is not actually the study of patterns. Both studies used student participants and (ignoring the effects of the changes) FS9 claims to have produced comparable findings. The only difference reported in FS9 was:

> *Compared to the reference experiment, the main difference is the improvement in time obtained with the web-based exercise compared to the paper based.*

which was quantified as "*the average reduction in time is 8 percent*." However, the author of FS9 also observed that he found the opposite results to those from FS8 for one of the tasks and the hypothesis related to time needed to perform the task. The authors of FS8 had noted that their results for this task did not support the hypothesis for instantiation $FS8_1$ and offered an explanation of why they thought this had occurred. This related to the use of paper-based solutions for this part of the study, making it difficult for participants to check correctness—whereas for $FS8_2$ Unix workstations were

used, allowing checking, (although there were also too few participants for analysis). Unfortunately, the author of FS9 does not offer any explanation for the difference that he observed.

- FS11 is a replication of FS7 (comparing the maintainability of designs developed with patterns to those developed without them). Both studies used professional software engineers, but for FS11 these used a software development environment rather than paper printouts. The conclusions of FS11 emphasized the need to assess patterns on an individual basis and found differences in the results for two patterns (Visitor and Observer). With regard to these two patterns the authors of the original study (FS7) observed:

A negative effect from unnecessary application of the Observer pattern, particularly for subjects with low pattern knowledge.

While for the replication (FS11) the authors noted:

Our results differ somewhat from those found by Prechelt et al. (2001) especially in the case of Visitor and Observer. While they found Visitor to be without significant harmful effects, few of our subjects achieved a good solution with it, even after the course. By contrast, we observed no significant harm done by using the Observer pattern

Neither FS9 nor FS11 can be considered to be *close* replications [42]. As is apt to occur with software engineering replications, both were *differentiated* replications, as the researchers made some changes to the form of the experiment. (Lindsay and Ehrenberg argue that close replications are needed first in order to establish consistency of results, with differentiated studies then helping to determine how widely these apply [42].) So, the lack of any close replications does make it more difficult to draw any firm conclusions about consistency. However, in both cases the replications used participants with similar backgrounds to those used in the originals (students for FS9 and FS8, software development professionals for FS11 and FS7). A possible explanation for the differences in the second pair may have arisen largely from the difference in the extent to which participants were familiar with patterns—which could have been greater for the professionals than would be likely to occur for students—but equally it could have arisen from the change in working mode, paper versus software tool.

The variations produced in these two replication studies do suggest that there is a need to perform close replications before differentiated ones. In both cases there was sufficient variation in the experimental conditions to explain the differences in the results, with no means of determining the actual cause.

4.4.2 Patterns Studied Most

For this group, we asked the question: "*Where the same pattern is studied in a number of experiments, do these reinforce one another or show different effects?*"

Three patterns have been studied rather more than others: *Composite*, *Observer*, and *Visitor*, largely in the same set of studies. However, in addressing their research questions, many of the papers focus upon the tasks involved in the studies, and it is not uncommon for these to involve more than one design pattern. Only the studies reported in FS5 (for FS8), FS7, and FS11 provided qualitative comments about the effects of individual patterns upon the participant's activities and solutions, and even for these, only limited information is available.

- For *Composite*, FS5 (for $FS8_1$ and $FS8_2$) considered its use as beneficial when used with Visitor (as noted above, many tasks involved multiple patterns, making it hard to separate their effects, with this being particularly true for *Composite*). However, in FS11, its dependence on knowledge about recursion created problems, which the authors considered might be because recursion was unfamiliar to the participants, while FS7 noted no specific effects.
- For *Observer*, FS11 noted no problems arising from its use (see quote above). The only comments in FS5 related to $FS8_1$ and concerned the benefits arising from the use of pattern-specific documentation. However, the replication in FS11 "did not find the negative effect that was observed in the original experiment" in terms of the expectation of easier implementation when provided with pattern documentation. FS7 observed that participants needed longer to understand the task involving this pattern, and that the greater complication introduced by its use was only justified where the increased flexibility was needed.
- For *Visitor*, FS5 noted that (when used with Composite) its use resulted in less time being taken ($FS8_2$) or led to fewer errors ($FS8_1$). In contrast, FS7 noted no specific effects, and that its presence did not increase the time needed to perform the tasks. FS11 considered it complicated, leading to longer development time and poor correctness, to the extent that participants avoided it. FS12 specifically set out to study this one pattern, using an eye tracker to determine how participants moved their attention around a class diagram. The authors concluded that it "*does not reduce the subjects' efforts for comprehension tasks*" but that where a UML diagram is provided in the canonical form for this pattern then it "*reduces the developer's efforts for modification tasks*."

Factors such as task size and background of participants may well have influenced these differences, and the extent of these does suggest that care is needed when using the outcomes from individual studies.

5 Triangulation with Experience

Given the limited set of (relatively rigorous) experimental studies, as well as the limited and sometimes differing conclusions from these about individual patterns, we decided to investigate whether other forms of empirical study might help resolve or explain these differences. To do this, we reviewed the set of papers that we had noted as containing "experience" during inclusion/exclusion. For this purpose we regarded an experience paper as being one that both provides a set of observations that are based upon

TABLE 9
Data Extraction Form Used for Experience Papers

	Element	Code	Interpretation
1	Reference No.	no	Our own code related to how the paper was found
2	Authors	no	Paper details recorded for convenience
3	Title	no	
4	Where published	no	
5	Are authors pattern developers?	yes	We tried to identify whether the authors had published tutorials or books about patterns, by looking at references
6	Form of study	no	How the experiences were observed
7	Type(s) of system	no	Details of any specific systems that provided the experience
8	Size of system(s)	no	Any indicators of size that were provided
9	Experience from own development or others	yes	Used to identify how direct the sources of experience were
10	Design/Coding	yes	The level of abstraction used in the paper
11	Development/Maintenance	yes	Where in the life-cycle the experience originated
12	Patterns Discussed		*(fields a–e repeated for each pattern discussed in the paper)*
	a. Pattern Name	no	Specific pattern for which experience was provided
	b. Pattern Source	no	Where the pattern was published
	c. Advantages/benefits	no	Positive experiences from using the pattern
	d. Problems/disadvantages	no	Negative experiences from using the pattern
	e. Conclusion	no	Any conclusions identified about use of this pattern
13	Conclusions	no	Any overall conclusions about using patterns
14	Form of link between experience & conclusions	no	How the conclusions (individual patterns or overall) were derived from specific experiences
15	Keep or reject	yes	The final recommendation

practical experience with using design patterns, and also where these observations are summarized as "lessons" that have explicitly been derived from the experience. We examined 22 candidate papers in detail, eventually selecting seven of these.

5.1 Data Extraction

As these papers are far less homogeneous than those describing experiments, in order to extract the data consistently we employed a data extraction form that was adapted from the one used for the experiments. The form (summarized in Table 9) consisted of three groups of questions. Where we provided specific codes for classifying an element, this is indicated; otherwise, entries were free text.

1. *Q1-4. Citation details.* Describing the source and how it was found.
2. *Q5-11. Study context.* These questions were used to identify the characteristics that might influence the way that any outcomes should be interpreted and weighted. We were interested in knowing how independent the authors were (by looking at the references to see if they were authors of patterns or books about patterns), in knowing about the type and size of system(s) that provided the source of the experiences, in knowing whether these experiences were first hand or not, in determining the level of abstraction at which the experiences were discussed (design or coding), and in establishing how these were related to the software life cycle.
3. *Q12-14. Information provided.* These addressed the details of the patterns involved, the conclusions about them, and how these were derived from the experiences.

For each paper this form was completed by both authors, working independently. Our conclusions were then put into a spreadsheet, checked for consistency, and then any differences were resolved. The final selection process was then based upon two main criteria:

- That the paper identified specific design patterns (necessary for our original purpose of triangulation with the results from the experiments).
- That the "lessons" described in a paper were linked to specific experiences.

For the second and third group of questions it proved difficult to answer every question for all papers due to the standards of reporting (we discuss this further in Section 6). For the second group (study context), this was made more problematical both because our chosen classification schemes proved to be a poor match and also because it often proved difficult to determine the right category from the available information (where one of us extracted a value and the other did not, we noted this as being a disagreement and then resolved it later). Examining the disagreements revealed that these were essentially differences of interpretation. This was far less of a problem for the third group (information provided).

5.2 Outcomes

Our expectation was that we might gain some qualitative assessments about individual patterns from these studies. These could then be combined with the qualitative elements that are provided in some of the reports on experiments (the triangulation element). However, a major limitation affecting almost all of the papers was the lack of any clear link between the experiences described in the paper and the conclusions (or "lessons") drawn by the authors. By failing to provide the cause-effect audit trail that is needed for any findings to be considered scientifically reliable, the authors made it impossible to assess the validity of their conclusions. In particular, this does mean the "expert assessment" elements of these reports are implicit rather than explicit. So the only papers that we retained for further analysis were

those where we judged that there was sufficient information provided about the linkage between cause and effect for us to have confidence in the assessment element.

In the next sections we summarize the set of papers that did provide some element of linkage between experiences with specific patterns and their conclusion. We then report on how well we were able to extract data from the wider set of candidate papers, identifying some of the reasons why this set was selected.

5.3 The Included Papers

To distinguish this group of papers, we will refer to them as **O1** to **O7** in the rest of this paper (using "O" for "observational"). The variety they exhibit makes it difficult to tabulate their characteristics in the same way as for the formal studies, so we have provided a brief summary of each one here.

1. **O1**. The focus is upon reuse in developing communications software [55]. A relatively early paper, and partly proselytizing in nature, it provides a long list of lessons but does link these (directly and indirectly) to the experiences gained from developing three systems. The paper concludes that using the *Reactor* pattern (a variant of *Observer*) has the benefit of increasing portability and avoiding the need for threads when handling events from multiple devices, offset by the loss of handler preemption as well as creating more complex flow of control to complicate debugging. As such it assesses the outcomes in terms of both benefits and disadvantages.
2. **O2**. Provides a description of experiences gained in developing an IDE [65]. Again, it is a relatively early paper, but it does specifically report on the use of three specific patterns (*Composite*, *State*, and *Abstract Factory*). However, their assessment that Composite eases the use of similar operations with different elements, while State simplifies the design of event responses, and Abstract Factory makes it easy to extend to new notations is not illustrated by any specific examples from the system they developed. The emphasis is also upon benefits, with any description of disadvantages being related to rather generic issues rather than the specific patterns.
3. **O3**. This report, by Wendorff, provides a valuable example of clear reporting of experiences [60]. Its focus is upon the experiences derived from maintenance rather than development, related to a large system that had been developed with patterns being used for many parts. It links the discussion of four particular patterns (*Proxy*, *Observer*, *Bridge*, and *Command*) to specific experiences with changing the structure of the system. The paper distinguishes between situations where the characteristics of specific patterns may be considered undesirable (such as the added indirection involved in using *Proxy*), as well as where they were used inappropriately (and why this was so), together with the consequences for the system maintenance.
4. **O4**. This report describes the experience of applying design patterns to construct a flexible system [43]. While the lessons are based upon specific patterns, they are not directly linked to them and we primarily retained this paper because of its contribution in terms of the experiences related to flexibility.
5. **O5**. The basis for this report was the experiences from developing a customisable diagram editor [63]. The authors discuss one architectural pattern (not used here) and six design patterns. For these, they assess their experiences in terms of the GoF claim that the insight from using patterns would "make your own designs more flexible, modular, reusable, and understandable," assessing the experience for each pattern in terms of those qualities. Their summary focuses on the different effects noted for behavioral and structural patterns.
6. **O6**. This study draws upon the industrial experience of one of the authors in order to assess three patterns (*Singleton*, *Abstract Factory*, and *Facade*) with the aim of assisting readers with recognizing potential pitfalls [16]. It particularly assesses the problems that can arise in the context of implementing these using Java, and its main limitation is that the industrial experience is distilled into abstract examples rather than cited directly.
7. **O7**. While at different points the authors describe this paper as an experiment, a quasi experiment, and a case study, it is largely observational in nature [14]. It reports on the experiences of 23 postgraduate students who were asked to develop two versions of a system (chosen from a range of topics) working either as individuals or in pairs. The first version was required to be without pattern use, and in the second they were asked to resolve any problems identified in the first by using at least two different patterns. The study notes that "students had difficulties in demonstrating exactly which problems have been solved by each pattern." There is little detailed analysis, but the authors did identify three issues as "most frequently reported by the students" that were resolved through the use of *Bridge*, *State*, and *Observer*. Unfortunately there is very little in the way of causal links in the reporting.

We now describe how well the different characteristics were reported in the experience papers we examined. While our discussion ranges across the full set of papers that were candidates for inclusion, we will particularly draw examples from the group above.

5.4 Describing the Source of Experiences (Q6-Q11)

Q6 ("form of study") proved difficult to code for this group of papers (experience papers are mostly observational, but organized in different ways), and most of the issues involved were essentially captured by the remaining questions. Hence, we did not separately analyze the results for Q6.

For Q7 ("type of system"), many papers did not report this, although it is quite important for the reader who might want to be reassured that the experience provided is relevant to their needs. Of the papers described in the

TABLE 10
Patterns Studied in Experiments and Experience Papers

Pattern	Scope	Formal Studies	Observational Studies
Creational Patterns			
Abstract Factory	Class	FS7, FS11	O2, O6
Factory Method	Object	FS2, FS4, FS13	
Singleton	Object	FS10	O6
Structural Patterns			
Adapter	Object	FS13	
Bridge	Object	FS10	O3, O5, O7
Composite	Object	FS4, FS7, FS8(2), FS9, FS10, FS11	O2
Decorator	Object	FS7, FS11, FS13	
Facade	Object	FS7	O5, O6
Behavioural Patterns			
Chain of Responsibility	Object	FS10	O5
Command	Object	FS4	O3
Iterator	Object		O5
Observer	Object	FS4, FS7, FS8(2), FS9, FS10, FS11	(O1), O3, O5, O7
Proxy	Object		O3
State	Object	FS3, FS4	O2, O7
Strategy	Object	FS13	
Template Method	Class	FS8(2), FS9	
Visitor	Object	FS7, FS8(2), FS9, FS10, FS11, FS12	O5

previous section, O1, O2, O3, and O5 did report the basic purpose of the systems involved (although Wendorff gave only a very vague indication). There may well be good commercial reasons for not giving much detail, but that should not prevent enough being given for general classification purposes. For O7 the range of project topics was listed.

Q8 was intended to extract the size of the system that formed the source of these experiences. Only four papers from those considered gave any indication of this, and then mostly indirectly. In addition, these four used a range of different measures, two of which were internal measures (KLOC, number of classes) while the third was an external measure (years of development plus number of developers). Of the set used in this study, O3 provided a single measure of size (1,000 KLOC) and O5 provided two measures: 50 KLOC and 173 classes. For O7 there was a range of systems produced, with average size around 800 LOC.

Q9 addressed the issue of whether the experience was from the authors' own development work or that of others (allowing for the possibility of being both). Apart from O7, authors appeared to largely report on their own experiences.

Q10 sought to distinguish between experience that was gained from working at a design level of abstraction or from using (code-oriented) realizations of patterns. Because these do not have clear boundaries, we sometimes disagreed in our interpretations.

Finally, Q11 focused on a rather important distinction, which was whether a paper described experiences gained from development of a system or from maintenance activities. While the former was by far the most common (unlike the experiments), if not always clearly identified, studies based on maintenance seem to provide a better retrospective view of how the use of patterns to structure a system can affect its form and performance.

Overall, while these characteristics proved to be useful in terms of providing the context for supporting the decision to include the set of papers O1-O7 described in Section 5.3, in many papers they were either poorly reported or not reported at all.

5.5 Describing the Experiences (Q12-Q14)

Few of the papers provided descriptions that were explicitly related to specific patterns and derived from their experiences regarding these. Even fewer provided descriptions that linked their experiences to their conclusions. The most positive example was the paper by Wendorff (O3), which did discuss specific patterns and provided some specific conclusions about these, illustrated by descriptions derived from the experiences. O5 and O6 also linked experiences to specific patterns, but in each case the experience itself was only described in the abstract.

5.6 Triangulation with the Experiments

Our original purpose was to identify if any patterns were studied using more than one form, and the extent to which these agreed. Table 10 lists the patterns studied in both the formal studies as well as the experience papers.

Table 10 indicates that the patterns which have the widest mix of study types, and hence which offer good opportunity for triangulation, are *Composite*, *Observer*, and *Visitor*. (For *Observer*, *O1* is in brackets as the paper reports upon a variation of Observer (namely *Reactor*), and so we cannot be sure that any comments about that will be applicable to *Observer*).

Overall, there was disappointingly little overlap between the sets of patterns investigated using the two forms. In particular, the results from experience that were reported most effectively (by Wendorff) were all for patterns that were only covered to a very limited degree in the experiments. In the rest of this section, we summarize the outcomes from the studies of the three patterns that have been examined most comprehensively: *Composite*, *Observer*, and *Visitor*.

TABLE 11
Summary of Qualitative Assessment for Composite

Paper	Participants	Qualitative Assessment
FS4	215 students	No pattern-specific comments
FS7	29 professionals	No issues identified (only used with other patterns)
$FS8_1$	74 students	No issues identified
$FS8_2$	22 students	No issues identified
FS9	28 students	No issues identified
FS10	15 students	No issues identified
FS11	44 professionals	The reliance of this pattern upon use of recursion was noted ("caused some problems") possibly because of an issue with knowledge of recursion
O2	n/a	No comments other than to indicate satisfactory use

5.6.1 Composite

Unfortunately, the available experience data for *Composite* lacks any examples that might make it possible to compare it with the results from the experimental papers (which also produce conflicting conclusions). Table 11 summarizes the different conclusions about this pattern.

In many studies the use of *Composite* went alongside that of other patterns, and while these sometimes attracted comments, *Composite* did not. Overall, we can assume that the use of *Composite*, while needing some knowledge of recursion, does not tend to create particular problems.

5.6.2 Observer

As noted earlier, there were somewhat differing conclusions about *Observer* from the experiments. Table 12 summarizes the qualitative data from the experiments (where available), plus that from the three experience papers.

TABLE 12
Summary of Qualitative Assessment for Observer

Paper	Participants	Qualitative Assessment
FS4	215 students	No pattern-specific comments
FS7	29 professionals	Its use increased the time needed for understanding and modifying the software when compared to a simpler solution, indicating a risk of overly-complicated structures arising from unnecessary use of this pattern (for the application)
$FS8_1$	74 students	When supported by pattern documentation, led to faster modifications
$FS8_2$	22 students	No specific comments
FS9	28 students	No specific comments (but note earlier observation about the different outcomes for this replication)
FS10	15 students	No specific comments
FS11	44 professionals	No problems reported and generally well understood
O1	n/a	Comments relate to event-driven systems only.
O3	n/a	One overly-complicated solution reported, created by a programmer who wanted to gain experience with using patterns.
O5	n/a	Increases flexibility of a solution and potentially eases reuse. Code becomes more understandable, but only for more expert users who know about the pattern.

TABLE 13
Summary of Qualitative Assessment for Visitor

Paper	Participants	Qualitative Assessment
FS7	29 professionals	Task involved an 'unnecessary Visitor', but effect of this not necessarily viewed as being harmful (data inconclusive)
$FS8_1$	74 students	When supported by pattern documentation, led to faster modifications
$FS8_2$	22 students	When supported by pattern documentation, led to fewer errors
FS9	28 students	No specific comments
FS10	15 students	No specific comments
FS11	44 professionals	Replication contradicts the original study (FS7) and indicates that its complexity led to increased time to perform tasks and lower correctness
FS12	24 students	Does not reduce effort for comprehension tasks, but may reduce effort for modification tasks
O5	n/a	Noted that "for somebody not acquainted with the Visitor pattern, the internal workings of the Visitor may be hard to understand"

From Table 12 we can see some benefits of triangulation, even if this is limited in scope. Both O3 and O5 provide some agreement with the results for *Observer* from the different experiments. The observations from O3 show some agreement with FS7, although providing no specific explanation of why this overcomplication might occur. O5 does, however, offer some explanation of the results from $FS8_1$ and FS11 in noting the likely distinction between the ease with which the two groups of participants were likely to be able to understand the code. There is also some small reinforcement from one survey [33], where understandability was seen as being decreased by the use of *Observer*.

5.6.3 Visitor

The results for *Visitor* are reported in Table 13. These are complicated by its being coupled with the use of *Composite* in many of the experiments that reported on its use (FS7, FS8, and FS11). (While the use of multiple patterns is realistic in the sense that pattern instances do interconnect within systems, from an experimental viewpoint it does make it harder to disaggregate the contributions of specific patterns.) FS8 provides little real information (the focus of the paper was on documentation), and as already observed, FS7 and FS11 provide conflicting results, with FS11 observing that despite thorough training: "most subjects either ignored the pattern or were confused by it" and also noting the poor correctness of solutions. FS12, which specifically studied this pattern, was ambivalent about its effect upon modification tasks. The one available observational study (O5) tends to agree with FS11 and FS12. Again, the survey of expert developers in [33] suggests a negative effect for understandability when using *Visitor*. Taken together, these results do at least partially indicate that designs based upon the use of *Visitor*, while likely to be harder to understand than nonpattern equivalents, may possibly be easier to modify

So, while we do have some examples of how observational studies may help with the interpretation of the results from experiments, we were unable to perform any significant degree of triangulation between the different empirical forms used in studying the other patterns. In part, this was largely because the poor reporting standards encountered resulted in our having too few usable data points, but equally because there were simply not enough studies available. In addition, most of the more formal studies were primarily concerned with examining issues such as documentation of patterns, rather than the effects of the patterns themselves.

We have drawn upon these experiences to propose a set of preliminary reporting standards for experience papers [12]. (Major headings for which we offer specific ideas are: title & ownership, structured abstract, keywords, introduction, background, lessons, threats to validity, conclusions & further work.) This is in the hope of encouraging better reporting since such papers clearly do have the potential to provide an explanation of experimental results, as well as to provide some assessment of how well they represent practice.

6 Discussion

6.1 Threats to Validity

For a secondary study such as this, we can identify three quite specific threats to validity that need to be considered:

- the effectiveness of the search process (internal),
- the selection and classification processes (internal),
- the completeness and the extent of the analysis as reported in the primary studies (external),

and we briefly examine each of these in turn.

For the search process, experience from conducting other studies of this form in software engineering suggests that a wide range of search engines should be employed since no search engine will find all of the relevant primary studies [10, Lesson 8]. Hence, we have used a broad set of search engines, backed up by a quite thorough manual search, as well as snowballing. So we can be reasonably confident that we have identified most of the relevant studies published in the computer science and software engineering literature.

The process of inclusion/exclusion was performed in two stages. The first, which consisted of excluding papers that were not about software design patterns and then categorizing these, was performed by one person. For the empirical papers, the subsequent process involved sifting first on title, then on abstract, and finally on the complete paper and was undertaken by both of us, working independently and then resolving any differences. Subsequent decisions about classification of the empirical papers proved to be rather more difficult, as demonstrated by the interrater measures: For some aspects, such as the form of a study, we achieved good agreement, but more subjective elements, such as the issue (from our predefined list) addressed by a paper, required joint discussion and resolution. Again though, since we have erred on the side of caution at each step, it seems unlikely that we have inappropriately excluded any material. We might also observe that for both selection and classification purposes, the abstracts provided were largely inadequate, making it necessary to read parts of the paper in order to make a decision (reinforcing [10, Lesson 10]).

The external validity of our findings in terms of aggregated knowledge about the patterns is influenced by the following two factors:

- No patterns have been studied very extensively;
- The experimental studies of OO patterns mostly involved tasks that required a mix of understanding and modification, rather than the development of new systems. So the influence of patterns upon development has only been studied for a few patterns.

One other aspect that should at least be noted is that most of the experimental papers are from two research groups, in Karlsruhe (with a preference for within-subject studies) and Hong Kong (preferring between-subject studies).

6.2 Answering the Research Questions

The answer to our first question ("which of the GoF patterns have been evaluated empirically?") is essentially summarized in Table 10. In all, 15 of the 23 patterns have been subjected to some form of formal empirical evaluation, albeit six of them (Singleton, Adapter, Bridge, Chain of Responsibility, Command, and Strategy) were only addressed in one study.

There was relatively little explanation in the studies that we found as to why particular patterns were chosen. This may perhaps have been because all of these studies were conducted by software engineers, with the choice of patterns used arising from a mix of opportunism (availability of suitable material) and their own experiences. Had they been conducted by cognitive scientists to explore particular design activities, then the choice might have been explained, but might have used patterns of less interest to software developers. Certainly, for all but one of the papers addressing the use of OO patterns, the choice was determined by the set of applications employed to provide tasks for the participants—and the selection of these seems to have been based on availability and the need for a tractable size so that participants could understand and change the code in the available time. In some cases, the choice was, of course, determined by the decision to perform a replication.

The only two studies that did choose a pattern for a specific reason were FS2, where the purpose was to study the use of the *Factory Method* pattern for API design, and FS12, which studied the *Visitor* pattern. Both of these were motivated by informal experiences of the researchers and, in the case of FS12, by differences in the outcomes of previous studies.

Formulating an answer to the second question ("what lessons about their use, and its effectiveness, are available from these empirical studies?") is constrained by the focus upon *maintenance* activities (understand and modify) in so many of the studies. While this is very relevant to the usefulness of patterns (and to their effectiveness), we are still left with little knowledge that might form useful lessons about their role in *development*.

This is arguably the consequence of the emphasis upon the use of laboratory experiments, a point that we return to below. Overall, therefore, we can currently draw little in the

TABLE 14
What the Studies Concluded about the Claimed Advantages

Advantages Claimed	Relevant Studies	Outcomes
Improved programmer productivity and program quality	FS2, FS3, FS4, FS12	"factories are demonstrably more difficult than constructors for programmers to use, regardless of context" [20] "The experiment suggested that the satisfaction of the pattern theme generally led to the conformance to the open-closed principle. However, three exception cases were found." [45] "utilizing deployed design patterns ... is found to be statistically significantly associated with the delivery of less faulty codes" [46] "(Visitor) does not reduce the subjects' efforts for comprehension tasks ... its canonical representation reduces the developers' efforts for modification tasks" [32]
Novices increase design skills	FS13, O7	"the main difficulty faced by novices was the incorporation of patterns into the initial class diagram" [31] "students found it difficult to relate the applied design patterns to specific design problems" [14]
Encourage best practices	FS2, FS7, FS11	"since many of the benefits of factories can be achieved by alternative solutions that do not incur the same usability penalty, the results ... suggest that such alternatives are often preferable to factories" [20] "unless there is a clear reason to prefer the simpler solution, it is probably wise to choose the flexibility provided by the design pattern solution because unexpected new requirements often occur" [51] "each design pattern tested has its own nature, so that it is not valid to characterize design patterns as useful or harmful in general (for maintenance)" [59]
Improved communication	FS8, FS9, FS10	"depending on the particular program, pattern comment lines in a program may considerably reduce the time required for a program change or may help improve the quality of the change" [52] "since our subjects were less expert and trained than [52], our support is stronger" [57] "team members can communicate more effectively with design pattern knowledge" [58]

way of conclusions about the conditions necessary for a given pattern to be successful (or unsuitable).

For the three patterns where there are enough studies for us to seek a more comprehensive answer to this question, Tables 11, 12, and 13 do offer some indication of how we can use observational studies to interpret the results from experiments—and there is also a clear warning about the risk of overly complicated solutions if a pattern is used inappropriately.

The third question ("what further research") poses a dilemma. Would it be better to augment the data available for those patterns that have already been studied, or instead to explore those patterns that either have limited coverage or are entirely unassessed? Our recommendation would be for the former—there is at least a baseline here, and some better reporting of experience could potentially create a more substantial corpus. Also, many of these patterns have been chosen on the basis of experience or because their use is quite widespread. Related to this is the question of the form of study that might be used—the experiences from this study and from the other studies reported in Table 15, discussed below, suggests that wider use of forms such as case studies might usefully be considered by researchers to allow deeper exploration, and that *close* replications are also needed.

It is also useful to look at the extent to which the formal studies address the four "claims" from [52] that we noted in Section 2.2. Table 14 summarizes how far these were addressed in the conclusions from each of the studies. From this we suggest that:

- There is reasonably good support for the claim that using patterns improves communication between developers and maintainers, at least when these are appropriately documented.
- There is no support for the claim that using patterns helps novices learn about design.
- In terms of their effect upon productivity and quality the results are ambivalent—arguably these issues are more strongly affected by the nature of individual patterns than the other two.

Overall, we can only conclude that what is clearly needed is more studies *about* patterns.

6.3 The Excluded Papers

There are two groups of studies that fall into the "empirical" category and that, while not meeting the criteria for inclusion, should be noted here as providing valuable knowledge about patterns and their use.

The first group is the surveys. We have already noted something about one of these that examined quality attributes for the *GoF* patterns [33], [34]. While the sample size used in this was small (20 developers), the paper does make a useful contribution in terms of identifying the aspects of patterns that matter to developers. A rather different form of survey is reported in [26], which involved

TABLE 15
Characteristics of Secondary Studies of Design Practices

Ref.	Topic	No. of primary studies used	Key Outcomes
[27]	Rational Unified Process	2 industrial case studies	Too complex for easy adoption and tailoring the RUP is also complex.
[44]	Model-driven engineering	4 industrial experience reports and 7 experiments	The experiments were mainly small scale, although one did perform a comparison that showed that MDE took longer but performed better.
[19]	Agile Methods	33 primary studies (mostly case studies, only 3 were experiments) and 3 secondary studies. 25 of the 33 primary studies were on XP.	Identified a number of benefits and limitations of agile methods, but the strength of evidence for these was very low.

analyzing design documents from 988 open-source projects. The author notes the lack of empirical data to support the claims for patterns and concentrates upon the *GoF* patterns alone. The analysis identifies how widely individual patterns were used across projects, but is otherwise focused upon developer behavior. As such, it can help future researchers to identify which patterns should be investigated from the perspective of how widely they are deployed. We might also note that, while it is not a survey as such, the literature review performed in [59] reports on the set of patterns found in the code that was analyzed.

This last paper provides a link into the second group of papers, which are based upon what we have termed "code analysis." These papers expand upon the ideas in two studies led by Bieman, using metrics to examine how the part of a design that is based upon design patterns evolves over time [7], [8]. Other papers in this group include [4], [5], [6], [22], [30], [47], [59]. While for our purposes these papers do not provide any pattern-specific data that could be included in our analysis, they do point the way toward making wider use of code analysis in studies of patterns.

6.4 Comparison with Other Studies of Design Practices

The secondary studies related to design that have been conducted so far are rather unevenly distributed across the knowledge headings used by the SWEBOK to categorize software engineering activities and knowledge [1]. For design activities, we can identify three studies that are clearly relevant. Two of them are primarily concerned with *adoption* rather than form of design practices: model-driven engineering [44] and the Rational Unified Process [27]. The remaining one comprises a study of what is known about agile methods [19]. However, this was effectively limited to reviewing studies of XP as there were only a few studies of other methods. Table 15 summarizes the main features of these three studies.

Looking at Table 15, a significant difference with this study is evident in the nature of the primary studies involved. For the three previous studies, many of the primary studies were case studies (however defined), with significantly less emphasis upon coding and/or laboratory experiments. Examining these a little more closely is quite informative.

The study of adoption of the Rational Unified Process [27] is a topic where the use of experiments might be considered less appropriate. Consequently, the authors designed their study protocol to address qualitative data, and identified a set of papers that seem to correspond well with those we have described as *experience*. The process of inclusion/exclusion is described very thoroughly, and we summarize the three key stages below:

- *Exclusion on title and abstract.* Their key criterion here was that "the study must present empirical data beyond anecdotal evidence," and resulted in reducing their initial set of 100 candidate papers to 36.
- *Exclusion on inspection of full text.* This involved making a quality judgment about the following issues and excluding any papers considered to be "insufficient" with respect to them.

  - a well-defined and limited study aim,
  - adequate description of the study method,
  - sufficient description of study context,
  - presentation of study effects,
  - a thorough analysis of results,
  - conclusions and answer related to the study aim.

  This had the effect of reducing the 36 candidates to five (5).
- *Exclusion on team review.* This again used the six criteria above, and resulted in the five studies being reduced to a final two.

Overall, the process is generally similar to that which we used, in demanding relative rigor from qualitative studies, and the effect on the final count is not dissimilar.

The study on Model-Driven Engineering [44] provides little detail about either the searching process, although this did seem to include snowballing, or the inclusion/exclusion process. Thirty-three papers were found, but eight (8) were then excluded as lacking any detail about the application concerned, leaving 25, although not all of these appear to have been used. Their third research question was generally similar to ours: "What evidence do we have on the impact of MDE on productivity and software quality?" The authors make the comment about data extraction that "it was not possible to extract information on the size of the projects for the majority of papers," again reflecting our experiences. Of their final set of papers, only three provided information that could be used to answer the third research question described above, with these consisting of: a company case study, an EU project with six small case studies and quasi experiments, and one report of the redevelopment of a legacy project.

The systematic review of agile methods [19] describes quite rigorous screening procedures and criteria for quality. They found more primary studies than expected (based on the three nonsystematic reviews that preceded it), although the focus of the primary studies was almost entirely upon XP. The case studies and surveys were predominantly undertaken in industrial settings. They noted that in many cases: "methods were not well described; issues of bias, validity, and reliability were not always addressed; and methods of data collection and analysis were often not explained well." Given the variation in the subject matter, it was perhaps not surprising that many of the primary studies were concerned with issues such as adoption. Of the four that did comparative studies of productivity, three found improvements from adopting XP, although the authors add a caveat to the effect that "none of these studies had an appropriate recruitment strategy to ensure an unbiased comparison."

When comparing our study with these, we can see many parallels: relatively small numbers of primary studies, quality issues with the primary studies, and poor reporting standards for observational studies.

6.5 Implications for Future Studies

One purpose for a mapping study such as this is to identify where further studies are needed and to help researchers determine how they might be organized. While the set of empirical studies reviewed in this paper provide a valuable contribution to our understanding of how and when design patterns might best be deployed, they also exhibit some significant limitations.

First, the emphasis upon the use of experiments clearly forms a limitation when studying design activities. Their use imposes a restriction upon both the depth (duration) of such studies and for within-subject forms in particular, they also limit the experience of participants (who are so often students). Table 6 emphasizes this quite clearly. Looking at Table 7, we might also question whether measures such as time to complete a task or the number of errors in the code are suitable measures to employ when studying design activities? **Recommendation 1:** *The use of experiments, particularly using student participants and short-term tasks, should be used with care for studies of design. Case studies may be more appropriate for exploring the complex cognitive issues involved [64].*

A second limitation is the basis for choosing which patterns to study. Only two studies set out to address the question of how a specific pattern was to be employed. There are some obvious issues here that it might be valuable to investigate, for example, whether (say) behavioral forms are more easily learned and used by novices than creational or structural ones—and the influence of scope? **Recommendation 2:** *Studies of design patterns should use research questions that are related to specific patterns and their roles.* (As in the example of [20].)

Third, and relating especially to the experience papers, reporting standards are simply too poor for the data to be reliably aggregated with that from more rigorous forms of study. While we have sought to use this experience positively by proposing a set of reporting guidelines [12], in this case it unfortunately provides little that is useful for the study of design patterns. **Recommendation 3:** *Observational studies need to be reported rigorously and ensure that the links between any conclusions and the reported experiences are explicit.* (See [12].)

Indeed, noting the potential that a survey offers for a more short-term aggregation of experience, we have undertaken a survey into experiences with patterns for software development, using a set of questions that are based upon the data extraction model that we used for the experience papers [66]. Our survey received 206 usable responses and identified a small number of *GoF* patterns as being highly valued (e.g., *Observer*), rather mixed views about another group (such as *Visitor*), and adverse opinions about another group (particularly *Flyweight*).

We would therefore suggest that the "gap" between the existing experimental and observational studies might be most effectively spanned by performing a set of case studies with experienced practitioners, spanning development to maintenance, and focusing upon a small number of the more controversial patterns (chiefly *Visitor*, *Singleton*, and *Facade*). Such studies could reveal more insight into the longer term consequences of design decisions regarding the use of patterns and upon their effect during maintenance.

7 Conclusions

Our mapping study has shown that, despite their prominence in software engineering, design patterns have not been subjected to other than limited empirical evaluation, and that much of this has also only been studying patterns indirectly. However, Table 14 indicates that there is some qualitative support for the value of patterns as an aid to maintenance (when documented as such), and also that patterns do not appear to help novices learn about design. Beyond that, the quality of the available studies generally proved inadequate for us to be able to identify any firm guidelines about when to use (or not to use) particular patterns, and more design-centric evidence is very much needed.

Design patterns can potentially provide the "knowledge schema" role described by Détienne [17]. However, the variety of form and scope that arises means that "blind" application of patterns with any sense of the potential limitations is unwise (as is illustrated by the example of the conclusions of FS2 that "the factory pattern erodes the usability of APIs in which it is used" [20]).

Our study indicates that we are currently far from having the necessary degree of knowledge for making evidence-based judgments about when to employ individual patterns. Hence, we hope that the recommendations will provide a valuable framework for future studies. A key role for a mapping study is to provide a baseline against which further research can be positioned to best effect. As such, the outcomes from this study provide some important pointers as to where this should be focused and how it might be organized.

Acknowledgments

This work was partly supported by an award from the UK Engineering & Physical Sciences Research Council (EP/E046983/1: Evidence-based Practices Informing Computing (EPIC)). The authors would like to acknowledge the help

received from Professor Barbara Kitchenham and they are very grateful to the anonymous reviewers for their observations and suggestions.

REFERENCES

[1] *Guide to the Software Engineering Body of Knowledge,* A. Abran, J.W. Moore, P. Bourque, and R. Dupuis, eds. IEEE Computer Society, 2004.

[2] B. Adelson and E. Soloway, "The Role of Domain Experience in Software Design," *IEEE Trans. Software Eng.,* vol. 11, no. 11, pp. 1351-1360, Nov. 1985.

[3] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language.* Oxford Univ. Press, 1977.

[4] L. Aversano, L. Cerulo, and M. Di Penta, "Relationship between Design Patterns Defects and Crosscutting Concern Scattering Degree: An Empirical Study," *IET Software,* vol. 3, no. 5, pp. 395-409, Oct. 2009.

[5] L. Aversano, G. Canfora, and L. Cerulo, "An Empirical Study on the Evolution of Design Patterns," *Proc. Sixth Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. The Foundations of Software Eng.,* pp. 385-394, 2007.

[6] L. Aversano, L. Cerulo, and M. Di Penta, "Relating the Evolution of Design Patterns and Crosscutting Concerns," *Proc. Seventh IEEE Int'l Working Conf. Source Code Analysis and Manipulation,* pp. 180-192, 2007.

[7] J.M. Bieman, G. Straw, H. Wang, P.W. Munger, and R.T. Alexander, "Design Patterns and Change Proneness: An Examination of Five Evolving Systems," *Proc. Ninth Int'l Software Metrics Symp.,* pp. 40-49, 2003.

[8] J.M. Bieman, D. Jain, and H.J. Yang, "OO Design Patterns, Design Structure, and Program Changes: An Industrial Case Study," *Proc. IEEE Int'l Conf. Software Maintenance,* pp. 580-589, 2001.

[9] O.P. Brereton, B.A. Kitchenham, D. Budgen, and Z. Li, "Using a Protocol Template for Case Study Planning," *Proc. 12th Int'l Conf. Evaluation and Assessment in Software Eng.,* 2008.

[10] O.P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, and M.A. Khalil, "Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain," *J. Systems & Software,* vol. 80, no. 4, pp. 571-583, 2007.

[11] D. Budgen, J. Bailey, M. Turner, B. Kitchenham, P. Brereton, and S. Charters, "Cross-Domain Investigation of Empirical Practices," *IET Software,* EASE special section, vol. 3, no. 5, pp. 410-421, Oct. 2009.

[12] D. Budgen and C. Zhang, "Preliminary Reporting Guidelines for Experience Papers," *Proc. 13th Int'l Conf. Evaluation and Assessment in Software Eng.,* pp. 1-10, 2009.

[13] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture.* Wiley, 1996.

[14] A. Chatzigeorgiou, N. Tsantalis, and I. Deligiannis, "An Empirical Study on Students' Ability to Comprehend Design Patterns," *Computers & Education,* vol. 51, pp. 1007-1016, 2008.

[15] E. Chung, J. Hong, M. Prabaker, J. Landay, and A. Liu, "Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing," *Proc. Conf. Designing Interactive Systems,* pp. 233-242, 2004.

[16] M.Ó. Cinnéide and P. Fagan, "Design Patterns: The Devils in the Detail," *Proc. Conf. Pattern Languages of Programs,* 2006.

[17] F. Détienne, *Software Design—Cognitive Aspects,* practitioner series. Springer, 2002.

[18] T. Dybå, B.A. Kitchenham, and M. Jørgensen, "Evidence-Based Software Engineering for Practitioners," *IEEE Software,* vol. 22, no. 1, pp. 58-65, Jan./Feb. 2005.

[19] M. Jørgensen, T. Dybå, and T. Dingsøyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information & Software Technology,* vol. 50, pp. 833-859, 2008.

[20] B. Ellis, J. Stylos, and B. Myers, "The Factory Pattern in API Design: A Usability Evaluation," *Proc. 29th Int'l Conf. Software Eng.,* pp. 302-311, 2007.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[22] M. Gattrell, S. Counsell, and T. Hall, "Design Patterns and Change Proneness: A Replication Using Proprietary Csharp Software," *Proc. 16th Working Conf. Reverse Eng.,* pp. 160-164, 2009.

[23] R.L. Glass, V. Ramesh, and I. Vessey, "An Analysis of Research in Computing Disciplines," *Comm. ACM,* vol. 47, pp. 89-94, June 2004.

[24] R.L. Glass, I. Vessey, and V. Ramesh, "Research in Software Engineering: An Analysis of the Literature," *Information & Software Technology,* vol. 44, pp. 491-506, 2002.

[25] R. Guindon, "Knowledge Exploited by Experts during Software System Design," *Int'l J. Man-Machine Studies,* vol. 33, pp. 279-304, 1990.

[26] M. Hahsler, "A Quantitative Study of the Adoption of Design Patterns by Open Source Software Developers," *Free/Open Source Software Development,* S. Koch, ed., Idea Group, Inc., 2004.

[27] G.K. Hanssen, F.O. Bjørnson, and H. Westerheim, "Tailoring and Introduction of the Rational Unified Process," *Proc. 14th European Conf. European Systems and Software Process Improvement,* pp. 7-18, 2007.

[28] B. Hayes-Roth and F. Hayes-Roth, "A Cognitive Model of Planning," *Cognitive Science,* vol. 3, no. 4, pp. 275-310, 1979.

[29] M. Höst and P. Runeson, "Checklists for Software Engineering Case Study Research," *Proc. First Int'l Symp. Empirical Software Eng. and Measurement,* pp. 479-481, 2007.

[30] C. Izurieta and J.M. Bieman, "How Software Designs Decay: A Pilot Study of Pattern Evolution," *Proc. First Int'l Symp. Empirical Software Eng. and Measurement,* pp. 459-461, 2007.

[31] M. Abdul Jalil and S.A. Mohd Noah, "The Difficulties of Using Design Patterns among Novices: An Exploratory Study," *Proc. Fifth Int'l Conf. Computational Science & Applications,* pp. 97-103, 2007.

[32] S. Jeanmart, Y.-G. Guéhéneuc, H. Sahraoui, and N. Habra, "Impact of the Visitor Pattern on Program Comprehension and Maintenance," *Proc. Third Int'l Symp. Empirical Software Eng. & Measurement,* pp. 69-78, 2009.

[33] F. Khomh and Y.-G. Guéhéneuc, "Perception and Reality: What Are Design Patterns Good For?" *Proc. 11th ECOOP Workshop Quantitative Approaches in Object Oriented Software Eng.,* p. 7, 2007.

[34] F. Khomh and Y. Guéhéneuc, "Do Design Patterns Impact Software Quality Positively?" *Proc. 12th European Conf. Software Maintenance and Reeng.,* pp. 274-278, 2008.

[35] B.A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-Based Software Engineering," *Proc. 26th Int'l Conf. Software Eng.,* pp. 273-281, 2004.

[36] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic Literature Reviews in Software Engineering—A Systematic Literature Review," *Information & Software Technology,* vol. 51, no. 1, pp. 7-15, 2009.

[37] B. Kitchenham, D. Budgen, P. Brereton, M. Turner, S. Charters, and S. Linkman, "Large-Scale Software Engineering Questions-Expert Opinion or Empirical Evidence?" *IET Software,* vol. 1, no. 5, pp. 161-171, Oct. 2007.

[38] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Review in Software Engineering," Technical Report EBSE 2007-001, Keele Univ. and Durham Univ. Joint Report, 2007.

[39] B. Kitchenham, R. Pretorius, D. Budgen, P. Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic Literature Reviews in Software Engineering—A Tertiary Study," *Information & Software Technology,* vol. 52, pp. 792-805, 2010.

[40] B.A. Kitchenham, D. Budgen, and O. Pearl Brereton, "Using Mapping Studies as the Basis for Further Research—A Participant-Observer Case Study," *Information & Software Technology,* special section from EASE, vol. 53, no. 4, pp. 638-651, 2011.

[41] C. Kohls and K. Scheiter, "The Relation between Design Patterns and Schema Theory," *Proc. 15th Conf. Pattern Languages of Programs,* pp. 1-14, 2008.

[42] R. Murray Lindsay and A.S.C. Ehrenberg, "The Design of Replicated Studies," *The Am. Statistician,* vol. 47, no. 3, pp. 217-228, 1993.

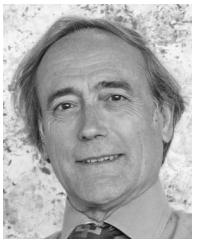
[43] G. Masuda, N. Sakamoto, and K. Ushijima, "Applying Design Patterns to Decision Tree Learning System," *Proc. Sixth ACM SIGSOFT Int'l Symp. Foundations of Software Eng.,* pp. 111-120, 1998.

[44] P. Mohagheghi and V. Dehlen, "Where Is the Proof?—A Review of Experiences from Applying MDE in Industry," *Proc. Fourth European Conf. Model Driven Architecture: Foundations and Applications,* pp. 432-443, 2008.

[45] 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," *Proc. Int'l Workshop Software Quality,* pp. 51-56, 2006.

[46] T.H. Ng, S.C. Cheung, W.K. Chan, and Y.T. Yu, "Do Maintainers Utilize Deployed Design Patterns Effectively?" *Proc. 29th Int'l Conf. Software Eng.*, pp. 168-177, 2007.
[47] M. Di Penta, L. Cerulo, Y.-G. Guéhéneuc, and G. Antoniol, "An Empirical Study of the Relationships between Design Patterns Roles and Class Change Proneness," *Proc. IEEE Int'l Conf. Software Maintenance*, pp. 217-226, 2008.
[48] M. Petticrew and H. Roberts, *Systematic Review in the Social Sciences: A Practical Guide.* Blackwell Publishing, 2006.
[49] L. Prechelt and B. Unger, "A Series of Controlled Experiments on Design Patterns: Methodology and Results," *Proc. Softwaretechnik '98*, 1998.
[50] L. Prechelt, B. Unger, and M. Philippsen, "Documenting Design Patterns in Code Eases Program Maintenance," *Proc. ICSE Workshop Process Modelling and Empirical Studies of Software Evolution*, pp. 72-76, 1997.
[51] L. Prechelt, B. Unger, W.F. Tichy, P. Brossler, and L.G. Votta, "A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions," *IEEE Trans. Software Eng.*, vol. 27, no. 12, pp. 1133-1144, Dec. 2001.
[52] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W.F. Tichy, "Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance," *IEEE Trans. Software Eng.*, vol. 28, no. 6, pp. 595-606, June 2002.
[53] R. Pretorius and D. Budgen, "A Mapping Study on Empirical Evidence Related to the Models and Forms Used in the UML," *Proc. ACM/IEEE Second Int'l Symp. Empirical Software Eng. and Measurement*, pp. 342-344, 2008.
[54] H.J. Rittel and M.M. Webber, "Planning Problems Are Wicked Problems," *Developments in Design Methodology*, N. Cross, ed., pp. 135-144, Wiley, 1984.
[55] D. Schmidt, "Using Design Patterns to Develop Reusable Object-Oriented Communication Software," *Comm. ACM*, vol. 38, no. 10, pp. 65-74, 1995.
[56] I. Sommerville, *Software Engineering*, eighth ed. Addison-Wesley, 2007.
[57] M. Torchiano, "Documenting Pattern Use in Java Programs," *Proc. Int'l Conf. Software Maintenance*, pp. 230-233, 2002.
[58] B. Unger and W. Tichy, "Do Design Patterns Improve Communication? An Experiment with Pair Design," *Proc. Int'l Workshop Empirical Studies of Software Maintenance*, pp. 1-5, 2000.
[59] 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 Eng.*, vol. 9, pp. 149-195, 2004.
[60] P. Wendorff, "Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project," *Proc. Fifth European Conf. Software Maintenance and Reeng.*, pp. 77-84, 2001.
[61] K.N. Whitley, "Visual Programming Languages and the Empirical Evidence For and Against," *J. Visual Languages and Computing*, vol. 8, pp. 109-142, 1997.
[62] L. Williams and A. Cockburn, "Agile Software Development: It's about Feedback and Change," *Computer*, vol. 36, no. 4, pp. 39-43, Apr. 2003.
[63] B. Wydaeghe, K. Verschaeve, B. Michiels, B. Van Damme, E. Archens, and V. Jonckers, "Building an OMT-Editor Using Design Patterns: An Experience Report," *Proc. Conf. Technology of Object-Oriented Languages*, 1998.
[64] R. Yin, *Case Study Research: Design & Methods*, third ed. Sage Books, 2003.
[65] W. Yuanhong, M. Hong, and S. Weizhong, "Experience Report: Using Design Patterns in the Development of JB System," *Proc. Technology of Object Oriented Languages and Systems*, pp. 159-165, 1997.
[66] C. Zhang and D. Budgen, "A Survey of Experienced User Perceptions about Design Patterns," submitted for publication, 2011.

Cheng Zhang received the MSc and PhD degrees from the University of Durham in 2006 and 2011, respectively. His research interests include the study of how software design patterns are used, empirical software engineering, and the use of evidence-based software engineering techniques.



David Budgen received the BSc and PhD degrees in theoretical physics from the University of Durham in 1969 and 1973, respectively. He is a professor of software engineering in the School of Engineering & Computing Sciences at the University of Durham, United Kingdom. His current research interests include software design, design of service-based systems, software development environments, and empirical software engineering, with a particular emphasis upon evidence-based software engineering. He is the author of *Software Design* (second edition, Pearson Addison Wesley) and of more than 100 refereed publications on software engineering topics. He is a member of the IEEE Computer Society, ACM, and IET.