RESEARCH-ARTICLE

# A Quasi-Experiment to Investigating the Impact of the Strategy Design Pattern on Maintainability

**GABRIEL COSTA SILVA**, Federal Technological University of Paraná, Curitiba, PR, Brazil

**VINÍCIUS CAMARGO ANDRADE**, Federal Technological University of Paraná, Curitiba, PR, Brazil

**REGINALDO RÉ**, Federal Technological University of Paraná, Curitiba, PR, Brazil

**RAFAEL MENESES**, State University of Maringá, Maringa, PR, Brazil



**Published:** 27 September 2021



SBES '21: Brazilian Symposium on Software Engineering
*September 27 - October 1, 2021*
*Joinville, Brazil*


https://doi.org/10.1145/3474624.3474636
ISBN: 9781450390613

# A Quasi-Experiment to Investigating the Impact of the Strategy Design Pattern on Maintainability

Gabriel Costa Silva
gabrielcosta@utfpr.edu.br
Department of Computing, Universidade Tecnológica Federal do Paraná
Cornélio Procópio, Paraná, Brazil

Vinícius Camargo Andrade
vcandrade@utfpr.edu.br
Department of Informatics, Universidade Tecnológica Federal do Paraná
Ponta Grossa, Paraná, Brazil

Reginaldo Ré
reginaldo@utfpr.edu.br
Department of Computing, Universidade Tecnológica Federal do Paraná
Campo Mourão, Paraná, Brazil

Rafael Cassolato de Meneses
rcassolato@gmail.com
Solutions Architect
Kansas City, USA

## ABSTRACT

GoF Design Patterns (DPs) are optimal solutions for several recurring problems in software development. As maintenance accounts for a massive amount of development costs, understanding the impact of DPs on maintainability is key to decide whether their benefits outweigh their costs. To test the hypothesis that the Strategy DP impacts maintainability, we carried out a quasi-experiment with 19 final-year undergraduate students. Participants performed enhancement tasks in two identical versions of Java classes that differed only in that a version implemented the Strategy DP whereas another version did not. As recommended in the literature, we used effectiveness as a representative proxy of maintainability and correctness as a measure of effectiveness. We found a statistically significant correctness change between treatments. Moreover, the effect size shows that a participant is 47.7 percentage points more likely to succeed when the Strategy DP is not used. Conversely, a participant is 5.5 times more likely to fail when the DP is introduced. Unlike most related studies, our findings suggest that the Strategy DP may reduce software maintainability although these findings need to be confirmed by independent and extended replications.

## CCS CONCEPTS

• **Software and its engineering → Software design engineering**.

## KEYWORDS

Software Maintenance and Evolution, Design Patterns, Software Architecture, Experimental Software Engineering

**ACM Reference Format:**
Gabriel Costa Silva, Vinícius Camargo Andrade, Reginaldo Ré, and Rafael Cassolato de Meneses. 2021. A Quasi-Experiment to Investigating the Impact of the Strategy Design Pattern on Maintainability. In *Brazilian Symposium on Software Engineering (SBES '21), September 27-October 1, 2021, Joinville, Brazil*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3474624.3474636


*SBES '21, September 27-October 1, 2021, Joinville, Brazil*

ACM ISBN 978-1-4503-9061-3/21/09…$15.00
https://doi.org/10.1145/3474624.3474636

## 1 INTRODUCTION

GoF patterns refer to a catalogue of 23 design patterns (DPs) presented by Gamma and colleagues in their seminal book [21]. These patterns have long been regarded as central players in solving design problems in software development [7, 28]. They extensively use best practices and quality principles of object-oriented design to providing optimal solutions for recurring problems [19, 49]. Practitioners encourage their use for a wide range of applications [10, 43, 45], and academics show growing interest in the topic [7].

However, literature reviews have revealed limited and contradictory evidence on the impact of DPs on maintainability [1, 3, 7, 37, 48]. For instance, a systematic review on the topic shows no experiment investigating the Strategy DP [3]. The lack of sufficient empirical evidence of whether DPs increase or decrease maintainability is critical since maintenance accounts for a massive amount of software development costs [42, 47]. If DPs decrease maintainability, their costs may outweigh their benefits.

Unlike several studies [3, 48], we focus on one single DP when conducting this present study. Our research question asks whether the Strategy DP significantly impact software maintainability. We do not envisage to provide a definitive answer in this paper, but to increase the knowledge of the field by providing a strong piece of evidence. This paper reports the first study of a family of experiments aiming to investigate the impact of DPs on maintainability.

When compared to related literature [1, 3, 7, 28, 36, 37, 48], the novelty of this study can be summarised as:

- Yielding the first evidence on the negative impact of the Strategy DP on maintainability;
- Carrying out the first study that uses experimentation to investigating the Strategy DP;
- Providing a complete package available online for replications and auditing – including versions of an application with and without the DP.

Our contributions are twofold. Firstly, our results differ significantly from directly related literature (Section 5). Therefore, changing the current understanding on the impact of the Strategy DP on

maintainability. Secondly, our study is one of a few experiments investigating the impact of DP on maintainability. According to Ampatzoglou et al. [3], there are only 12 studies with similar purpose to ours, of which only 4 are experiments. Therefore, we are not only increasing the body of literature in this field, but also making possible the use of methods for aggregating results [35].

To testing the hypothesis that the Strategy DP impacts software maintainability, we used Wohlin's framework to conduct a quasi-experiment[1] with 19 final-year undergraduate students in software engineering (*participants*). Participants ($n$=19) performed enhancement *tasks* in two identical versions of Java classes (*object*) that differed only in that a version implemented the Strategy DP (*experimental treatment*) whereas another version did not (*control treatment*). Like similar studies, we used effectiveness as a representative proxy of maintainability (*dependent variable*) and correctness as a measure of effectiveness (Section 3).

We found a statistically significant correctness change between treatments ($p$-value = 0.003, $\alpha$ = 0.05). Moreover, our analysis of the effect size shows that a participant is 47.7pp more likely to succeed when the Strategy DP is not used. Conversely, a participant is 5.5 times more likely to fail when the DP is introduced (odds ratio = 10.8) (Section 4).

Next, we analyse our study compared with related work, showing that our study contributes to enrich the body of literature on DPs evaluation by presenting results that differ from most results in the literature, and by rigorously synthesising concepts and techniques from DPs, software quality, maintenance, architecture and empirical research (Section 5). Based on the analysis of our findings, we reject our null hypothesis and discuss the implications of our results for our research question (Section 6).

Finally, we used guidelines from Shadish et al. [38] to carry out several follow-up analyses investigating the validity of our inferences (Section 7). Section 2 explains the Strategy DP, whereas Section 8 concludes the paper.

## 2 BACKGROUND

A software pattern is a reusable solution for recurring software problems [3, 37]. These patterns can be grouped into catalogues for several kinds of problems, such as architectural [12], integration [22], enterprise application [2], and DPs [21]. GoF DPs [21] is the most popular catalogue of software DPs [37]. GoF is an acronym for "Gang of Four", which is a reference to the four authors of the book that popularised DPs [3]. The GoF catalogue synthesises best practices and quality principles of object-oriented design in 23 patterns [19, 49].

The Strategy pattern is one of the 23 patterns described in the GoF book. It enables the interchangeable use of a family of algorithms. Its main advantage is that algorithms can vary independently from clients using it [21]. Figure 1 presents the original example for the Strategy pattern used in the GoF book.

Figure 1 shows that *Context* holds a reference (*strategy*) to an abstract class or interface – the *Strategy*. The *Strategy* defines a family of algorithms, represented by *ConcreteStrategyA*, *ConcreteStrategyB*, and *ConcreteStrategyC*. Clients interested in using one of the implementations of the *Strategy* calls the *ContextInterface()* method, responsible for executing the *AlgorithmInteface()* method in the concrete implementation. The client calling the *ContextInterface()* method, in the *Context* class, can change the algorithm used just by indicating the concrete strategy they want to use at runtime.

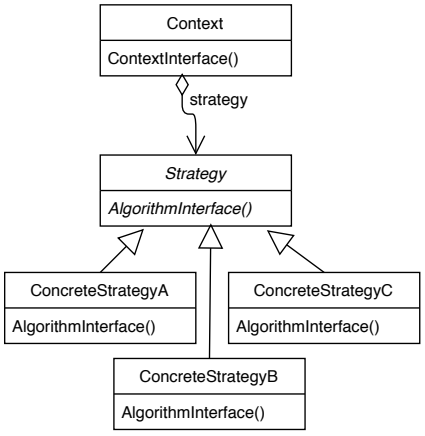


**Figure 1: The Strategy DP, as presented by Gamma et al. [21].**

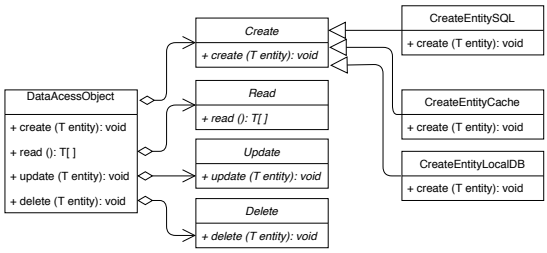


**Figure 2: Strategy pattern for data access management. Different data sources can be used interchangeably.**

The Strategy pattern prioritises relationships of *has-one* instead of *is-one* by holding a reference to an abstract class instead of using inheritance directly. Defining an abstract strategy, the pattern enforces a contract to be fulfilled by concrete strategies. The Strategy DP separates code that varies by defining a family of algorithms. Finally, the pattern increases software modularity by encapsulating strategies in units interchangeable at runtime. All these practices are considered good object-oriented design principles [19, 21, 36].

The Strategy DP can be used to solve several problems. In this study, we concentrate on the problem of managing different data sources at runtime. Figure 2 shows an example of a typical data access management (DAO) scenario. The *DataAccessObject* class (representing *Context*) provides the traditional CRUD operations (representing *ContextInterface* methods). Each CRUD operation is linked to methods (representing *AlgorithmInterface* method) in abstract classes (representing the *Strategy*) that define the contract for each operation. Concrete implementations of each operation (representing *ConcreteStrategy*) define how operations will perform in different data sources. The client transparently chooses the data source at runtime.

[1]Throughout this paper, when referring to the empirical method we used, we prefer "experiment" instead of "quasi-experiment" for readability reasons.

# 3 STUDY PLAN & EXECUTION

This quasi-experiment aims to analyse the Strategy DP for evaluating its impact on software maintainability. This study is undertaken in the context of undergraduate students performing maintenance tasks in Java classes with and without the Strategy pattern.

Our study is mainly based on the framework of Wohlin et al. [46], as it enables rigorous experiment preparation and execution. Moreover, we followed several recommendations [25, 30, 40] to make it possible to replicate this study. The next sections detail the experiment protocol and its execution whereas Section 4 presents the results of our hypothesis testing and follow-up analyses.

## 3.1 Participant Selection

*Sample size.* To ensure sufficient statistical power [33], we used power analysis to set the appropriate sample size for this experiment, as recommended by Ellis [16]. To do so, we needed historical data for effect size estimation and the expected power level for this experiment [11]. We set power level to the traditional 0.8 [14], as we had no reason to change this value. As related literature does not show data for effect size estimation [1, 37, 48], we used data from our pilot experiment (Section 3.7), as suggested by Ellis [16].

In our pilot experiment, 50% of our participants succeeded when maintaining classes without the DP, but failed when the DP was introduced. We observed no case for the opposite situation. Taking this result into account for a two-tailed McNemar test, paired data, power level of 0.8 and $\alpha = 0.05$, the recommended sample size is 13 participants. Therefore, our current sample of 19 participants is sufficient for the purpose of this study.

*Recruitment.* To recruit participants, we advertised the experiment for software engineering students enrolled in the software architecture module at the university where the first author works. The first author was the lecturer responsible for this module, which facilitated recruitment of participants. As usual [13, 41], student selection was driven by convenience, according to the student interest in participating.

A group of 19 final-year undergraduate students in software engineering took part in this study as participants. Students have been widely accepted as participants in SE studies as they [6, 13, 23, 41, 44, 46]: (i) are of easy access for academics, (ii) can be handful when testing initial hypotheses, and (iii) may be representative of junior professionals under certain conditions [23, 41]. Moreover, students are common in related studies [1, 37, 48].

*Ethical considerations.* For ethical reasons, the participation in this study was voluntary. To motivate students' participation, each student received a bonus in their final mark as a reward for their participation. Thus, students that did not take part in this study did not suffer any negative impact in their marks. Assigning credits as incentive is common in DP studies with students [37].

Since we are dealing with students, we thoroughly followed guidelines proposed by Carver and colleagues [13] to ensure both research and pedagogical value in this study. We implemented the guidelines as follows:

- *Ensured adequate integration of the study into the course topics* by discussing with students the current gap on understanding the impact of DP on software quality from an empirical perspective. We used several literature reviews [3, 7, 37, 48] to supporting our discussion;
- *Integrated the study timeline with the course schedule* by undertaking the study when DP was the subject topic in the module;
- *Reused artefacts and tools as appropriate* by using Java classes as objects since they are the focus of the taught module;
- *Wrote a protocol and had it reviewed*;
- *Obtained students' permission for their participation in the study* by requesting the participant consent before starting the experiment;
- *Set participants' expectations* by explaining the goals and the protocol for the experiment;
- *Documented information about the experimental context in detail* by writing down a protocol before starting and this paper after completing the study;
- *Implemented policies for controlling/monitoring the experimental variables* by using a set of online forms to collect data;
- *Planned follow-up activities* by giving students an overview on how to analyse the results, discussing the main threats with students, and setting up the next steps;
- *Built a lab package* by writing this paper and making the instruments used available online.

## 3.2 Selection of Variables

*Independent variable.* The approach used for implementing Java classes is our independent variable. This variable has two possible values (i.e., treatments): the traditional way that does not use the Strategy DP (i.e., control treatment), and the use of the Strategy DP (i.e., experimental treatment). The independent variable is measured in a categorical scale, whereby each treatment represents a category.

Although most studies on DP have been using arbitrary patterns as object of study [48], we chose the Strategy DP because: (i) it has been investigated only in 22% of papers reporting studies on the impact of DPs on software quality [3]; (ii) developers have only positive views on the Strategy pattern [49]; (iii) it has recently been recommended for several use cases [4, 20, 43, 45]; and (iv) our early interest lies on behavioural patterns.

*Dependent variable.* We are fully aware that several maintainability definitions have been used in software maintenance, quality and DP studies. However, the ISO/IEC 25010 drives our study as it is the definition most often used for investigating software quality attributes [34]. The ISO/IEC 25010[2] defines maintainability as *"the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements."*

As recommended by Kitchenham et al. [29], we avoided multiple outcome measures in this study, focusing only on effectiveness as our dependent variable. We measured effectiveness as the number of tasks correctly performed (correctness) divided by the total of tasks performed. "Correctly performed" means that participants performed tasks without any coding error.

As a proxy of effectiveness, correctness is a measure less often used in maintainability studies [24], but often used in DP studies [37, 48]. Correctness is a categorical variable, whereby valid values

[2] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

**Table 1: DAO classes used in our study.**

| # | DAO Class | # of lines of code | # of methods | Coupling (CBO) | Complexity (WMC) | Cohesion (LCOM) | # of Participants Assigned |
|---|---|---|---|---|---|---|---|
| 1 | AnexoDAO | 117 | 5 | 2 | 33 | 10 | 4 |
| 2 | AtaParticipanteDAO | 122 | 5 | 2 | 34 | 10 | 4 |
| 3 | CampusDAO | 185 | 7 | 2 | 55 | 21 | 1 |
| 4 | ComentarioDAO | 133 | 5 | 2 | 36 | 10 | 6 |
| 5 | PautaDAO | 115 | 5 | 2 | 33 | 10 | 4 |

are "yes" or "no", meaning the correctness on performing the task. Therefore, we did not consider partial success.

## 3.3 Task

To evaluating maintainability, participants had to perform maintenance tasks. Our participants performed one single maintenance task to prevent fatigue [38]. The task consists of changing a Java class by implementing a new method. Details are available on our online experimental package. Requiring modification tasks is common in DP studies that evaluate software quality [37, 48].

## 3.4 Hypothesis Definition & Study Design

The null hypothesis states that the number of correctly performed maintenance tasks does not change significantly when the Strategy DP is introduced. The alternative hypothesis states the opposite.

Although fully randomised experiments may mitigate the feasibility of alternative explanations for effects [38], we decided for a within-subject design because [38, 46]: (i) it is important from a pedagogical perspective to understand the student's perception of value before and after the introduction of the DP; (ii) it requires less resources; (iii) we could not isolate participants; (iv) we could not prevent that a student had preference for a treatment; and (v) this design is common in DP studies [48]. Therefore, all participants worked on both control and experimental treatments (paired data).

Since the within-subject design may cause several negative effects, we randomly sorted treatment order before assigning them to participants to increasing experimental validity [38, 46]. In addition, participants performed a pretest as usual in SE quasi-experiments [26]. A pretest is like a task (post-test), but its outcome is not analysed along with other tasks. Instead, pretests are used for follow-up analyses, like to check for testing effects [38, 46]. We used one pretest only to mitigate the chance of fatiguing participants.

## 3.5 Instrumentation

Java classes from a software system used in our university[3] were handled as objects in this study. This is an application used by our university staff for fulfilling one of its internal processes. The choice for such a system was driven by its: (i) source code availability, (ii) short classes, and (iii) use of familiar technologies to our participants. Authors of this study are neither users nor developers of the application used.

To enable analysing control and experimental treatments, we need two versions of the same software application. Whereas one version would use the DP, the other would not. Although several applications have been used in the literature, they do not provide two identical versions differing only by the use of the DP [48].

Therefore, we followed guidelines from Kerievsky [27] to refactoring 5 DAO classes (Table 1) in order to use the Strategy DP. DAO classes because they are a good fit for the Strategy pattern (Section 2). Moreover, DAO classes are well understood by our participants as they work with DAO classes in several prior modules.

Original and refactored classes (in pairs) where randomly assigned to participants to mitigate selection bias [38]. Thus, the number of participants associated with each DAO class is heterogeneous, as Table 1 shows. As Shadish et al. [38] highlight, this is expected for any randomisation.

Other relevant instruments used in this study are:

- *Checklists* used as guidelines to assist participants when performing tasks;
- *Feedback form* used as a measurement instrument to evaluate participants' perspectives;
- *Characterisation questionnaire* used as a self-assessment instrument to identify participants' characteristics, such as age and gender;
- *Development tools* used to (i) clone code from the Github repository, and (ii) edit and run the code.

All instruments used in this study are available online[4] to encourage future replications.

## 3.6 Data Analysis

The analysis of our results is mainly based on recommendations from Ellis [16]. In particular, we concentrate on the comparison between groups and the practical significance of our results. Therefore, we analyse our results by using the: i) difference between treatment probabilities, i.e., percentage points unit (*pp*); (ii) relative risk (rr) of failing in the maintenance task; and (iii) odds ratio. In addition, charts and tables are used to supporting data visualisation.

The Type I error, also known as significance level ($\alpha$), was set to traditional 0.05 [31] as we had no reason to increase or decrease this value. This $\alpha$ level implies on a confidence level of 0.95 when accepting the null hypothesis. The McNemar test was used to identify whether there is a statistically significant correctness change between treatments. We used McNemar test as it is [18]: (i) in line with our hypothesis; (ii) intended for categorical data; and (iii) appropriate for paired data.

[3] https://github.com/utfpr-dv/sireata.git

[4] https://github.com/EmpiricalStudies/strategy-dp-dao.git

Andrews et al. [5] underpinned our inferential statistic test selection whereas Field et al. [18] and Ellis [16] provided foundation for statistics and effect size measures we used. We used the R statistical software[5] and several of its packages for performing data analyses. Our data sets and R scripts are available online to enable replications and auditing.

### 3.7 Experiment Execution

A pilot experiment was performed to check the task complexity, experimental instruments, and time required for performing the tasks. The pilot experiment was undertaken about six months before the official experiment by a group of eight undergraduate students that did not take part in the official experiment. The result of the pilot experiment prompted some changes, leading to the current protocol detailed in this paper. In addition, it provided preliminary guesses on the results.

The experiment was undertaken in five steps. Training was the first step. Performed as part of the taught module, students were introduced to the concepts of DPs and implemented several GoF DPs. Furthermore, we discussed the current state-of-the-art in DPs and their gaps. The Strategy DP received special emphasis as it would be the subject of our experiment. Material used for training in the module mainly consisted of the GoF book [21] and websites[6]. The total training time in the classroom was of 12 hours, delivered over four weeks. In addition, students had home assignments.

Briefing was the second step. In the first meeting of the module, we presented the experiment to students, without giving any detailed information on when or how it would be performed. After the training was completed, we handed out an introduction script to students. This script provides an overview of the experiment, its purpose, the task they should perform, and benefits of participating. We discussed the benefits of participating, including the reward and the contribution to the software engineering community. Students were given one day to decide whether they would like to take part.

Next, we performed the participant selection. Those who accepted taking part in the experiment were asked to sign in the participant consent form, consenting in taking part in the experiment. Each participant randomly picked a unique number from a hat, which was used to identify them throughout the experiment.

The number was secret to the lecturer/experimenter to preventing bias towards better students. In addition, we asked them to fill out the participant characterisation form. This form collected information regarding the participants' background, such as professional experience. Questions in this form were mainly based on guidelines proposed by Feigenspan et al. [17].

Then, we carried out the experiment. The experiment consisted of performing the experimental task as defined in Section 3.3 by using the instruments defined in Section 3.5. Participants had to clone the application code from a given repository, open it on an IDE, and perform the required task. Most participants used computers available in our classroom (laboratory) although they were free to use their own laptops if their preferred. Participants could run, edit and test the code at their will.

[5]http://www.r-project.org
[6] https://refactoring.guru/

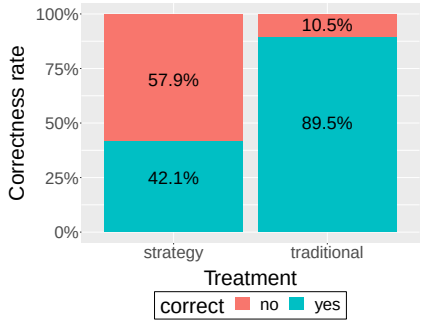


**Figure 3: Correctness differences between treatments.**

During the experiment execution, participants were free to get in and out of the classroom, e.g. go to the toilet. The experimenter observed the experiment execution without interfering in tasks performed by participants. The experiment was performed in two hours during regular class hours.

Finally, participants provided their feedback. When the experiment had finished, participants were asked to fill out a feedback form, assessing their experience in this experiment. The experimenter also provided a feedback in the first class after the experiment execution, by analysing and discussing results with students.

## 4 RESULTS

This section shows the results of our hypothesis test. In addition, this section presents follow-up analyses used in Section 7 to analyse whether internal and external factors may threaten our conclusions. To prevent the familywise error problem [18, 29, 38], we did not carry out any statistical significance tests in follow-up analyses. Section 6 discusses the implications of our findings.

### 4.1 Hypothesis Testing

The stacked bar chart in Figure 3 reveals that the likelihood of succeeding in the maintenance task is 47.7pp higher when the DP is not used. In addition, we observe that a participant is 5.5 times more likely to fail when using the DP compared with the traditional DAO implementation (i.e., the relative risk). Taking the odds ratio into account, the odds of failing when using the Strategy pattern is 10.8 times higher than when not using it ($1.77 \leq 95\%$ CI $\leq 123.74$).

The contingency table in Table 2 shows that 2 participants (10.5%) failed whereas 8 participants (42.1%) succeeded in the maintenance task, regardless of the treatment. On the other hand, we observe that 9 participants (47.3%) performed the maintenance task correctly only when the traditional implementation was used. The table shows no occurrence for the opposite situation.

Taking results of Table 2 into consideration, the McNemar test shows a statistically significant correctness change between treatments ($p$-value = 0.003), suggesting the difference between treatments is unlikely to be achieved by chance.

### 4.2 Employment Analysis

Figure 4 shows two charts that compare the correctness of treatments according to the employment status. The relative risk of

**Table 2: Contingency table.**

| Treatments (Correct?) | | Traditional No | Traditional Yes |
|---|---|---|---|
| Strategy | No | 2 | 9 |
| | Yes | 0 | 8 |

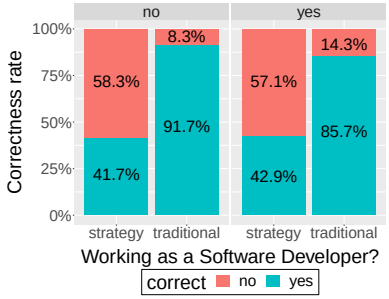


**Figure 4: Follow-up analysis: employment status.**

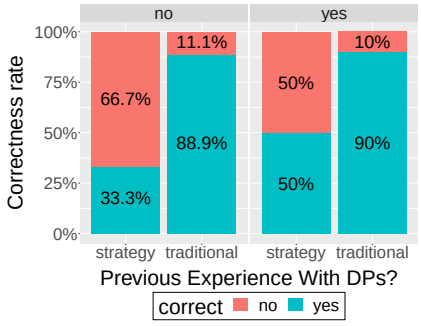


**Figure 5: Follow-up analysis: previous DP experience.**

failing when the DP is introduced for participants working as software developers (rr = 3.99) is about 1.5 less than the overall result, whereas for other participants (rr = 7) the relative risk is about 1.5 greater than the overall result. Although this result shows that software developers performed better than other participants, it does not suggest a significant change in the overall result.

## 4.3 Previous Experience Analysis

Like for the employment analysis, the relative risk of failing when the DP is introduced slightly differs (Figure 5). The relative risk for participants with previous experience on DPs (rr = 5) is about 0.5 less than the overall result, whereas for other participants (rr = 6) the relative risk is about 0.5 greater than the overall result. Again, this result shows that previous experience with DPs can lead to better results, but it does not change significantly the overall result.

## 4.4 Treatment Order Analysis

Figure 6 shows two charts that compare the correctness of treatments when the experimental treatment was introduced first and latter. According to the chart, the likelihood of succeeding in the maintenance task for the Strategy pattern is 29.5 pp higher when

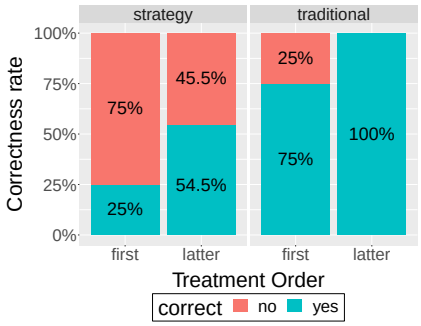


**Figure 6: Follow-up analysis. The order the pattern is introduced matters.**

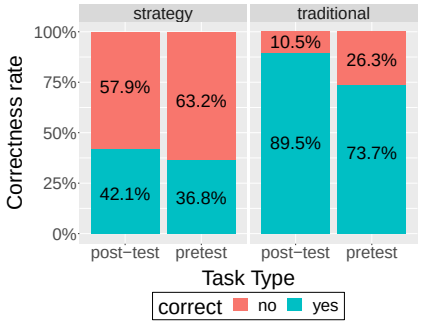


**Figure 7: Follow-up analysis: pretest and post-test.**

the pattern is introduced latter. Conversely, the relative risk of failing when the pattern is introduced first is 1.6 times higher than when it is introduced latter.

Unlike the result for the Strategy, the second set of charts shows that the correctness decreases with time for the traditional implementation. It is important to notice that the chart takes into account the order with which the Strategy pattern was introduced. Therefore, the correctness when the traditional implementation was applied first (100%) was 25 pp higher than when it was applied latter (75%), and the relative risk of failing is higher when the traditional implementation is applied latter.

## 4.5 Testing Analysis

Figure 7 shows two charts that compare the correctness achieved for both treatments during pretest and post-test. The likelihood of succeeding in the maintenance task for the Strategy pattern is 5.3pp higher in the post-test. Conversely, the relative risk of failing in the pretest is 1.09 times higher than in the post-test. This result suggests that the correctness increases in the post-test.

Like the result for the Strategy pattern, the chart shows that the correctness increases in the post-test for the traditional implementation. However, unlike the Strategy pattern, the correctness for the traditional implementation increases significantly (15.8pp). Conversely, the relative risk of failing in the pretest is 2.5 times higher than in the post-test.

### 4.6 Perception of Participants

As Section 3.5 presents, we used a feedback form to collect the perceptions of participants after the experiment. This section summarises participants' perceptions as identified by their feedback.

In a categorical scale whereby zero means very hard, and five means very easy, most participants considered performed tasks as easy (ease $\geq 3$), apart from two participants (10.5%). These two participants disagree in their perceptions, age, gender, and previous DP experience. Most participants (73.7%) found that the control treatment was the easiest to modify whereas only three participants (15.8%) found that the experimental treatment was the easiest, and two participants (10.5%) found no difference in the ease of modifying treatments. The three participants who found the experimental treatment the easiest to modify disagree in their perception on age, gender and previous DP experience. The same is true for those two participants who found no difference between treatments, apart from previous DP experience. Finally, most participants (89.4%) found that using the Strategy DP is beneficial to DAO classes.

## 5 RELATED WORK

Recently, several literature reviews summarising the state-of-the-art on software patterns have been published, each with different focus. Whereas Zhang and Budgen [48], Ali and Elish [1] and Ampatzoglou et al. [3] concentrate only on GoF DPs, Bafandeh Mayvan et al. [7] extend their focus to DPs, and Riaz et al. [37] cover software patterns in general.

Although we extensively used these reviews to ensure that our study design is inline with current practice (Section 3), our results mainly differ from that found in related literature. Whereas our results suggest that the Strategy DP may reduce maintainability (Section 6.2), Zhang and Budgen [48] report qualitative evidence that DPs may support maintenance, and Ampatzoglou et al. [3] analyse that 78.3% of DPs positively impact maintainability.

On the other hand, Ali and Elish [1] show that 54% of studies on maintainability presented negative impact whereas 23% presented positive, and 23% presented neutral impact. The impact changes when other quality attributes (QA) are analysed [1, 3]. As Ampatzoglou et al. [3] conclude, *"(...) the reported research to date on the effect of GoF patterns on software quality attributes are controversial; because some studies identify one pattern's effect as beneficial whereas others report the same pattern's effect as harmful."*

Among literature reviews, Ampatzoglou et al. [3] summarise studies relating GoF patterns and the QA focus of study. Therefore, we used their summary to finding studies directly related to ours. This enables analysing how our study compares to directly related work. Ampatzoglou et al. [3] show two studies investigating the impact of the Strategy pattern on software maintainability – both presenting positive impact.

Rajan et al. [36] mainly differ from ours in that our is an empirical investigation whereas their work consists of a proposal to increase modularity by adapting GoF patterns in order to achieve concurrency in software systems. They conclude that the pattern has positive impact on modularity since it properly enabled the encapsulation of concurrency and synchronisation concerns [36].

Therefore, different results between ours and Rajan's study may be explained by significant differences in purpose, design, and object used in both studies. Furthermore, we used the ISO/IEC 25010 definition of maintainability for our study whereas Rajan and colleagues used an arbitrary definition of modularity [36], which is a sub-characteristic of maintainability in the ISO/IEC 25010.

More similar to ours is the study conducted in Khomh and Gueheneuce [28]. Although we share similar purpose – to investigate the impact of GoF DPs on maintainability – and overall result – that GoF DPs may reduce maintainability, our studies differ significantly in design and in results obtained for the Strategy pattern.

Whereas we used experimentation, Khomh and Gueheneuce [28] used survey as their empirical method. Unlike our study, Khomh and Gueheneuce [28] used software engineering professionals rather than students as participants. Instead of focusing only on a single QA, Khomh and Gueheneuce [28] used several QAs. Like our study, Khomh and Gueheneuce [28] collected responses from a small sample of 20 participants. In addition, other study design differences can be explained by the empirical methods used in both studies.

Finally, whereas we found that the Strategy pattern negatively impacts one aspect of maintainability, Khomh and Gueheneuce [28] found mainly positive impact for the Strategy pattern on the QAs they investigated. This difference may be partially explained by methods and designs used, but also by the type of participants. It is interesting to highlight that Zhang and Budgen [49] also performed a survey research with professionals, similar to Khomh and Gueheneuce [28]. Zhang and Budgen [49] also found only positive perspectives of developers for the Strategy pattern. Comparing our results with those obtained by Khomh and Gueheneuce [28] and Zhang and Budgen [49], one could suggest that the use of professionals in our study could significantly change the results, although we did not find sufficient evidence for this claim in our analyses, as we discuss in Section 7.1.1. Perhaps, the good perceptions professionals have on the Strategy pattern make them overlook the pitfalls of applying the pattern. Although we noted the study of Zhang and Budgen [49], their study is not directly comparable to ours. They do not take any QA into account, but the usefulness of each GoF pattern instead.

## 6 DISCUSSION

This section discusses the impact of findings in Section 4 whereas the next section analyses threats to the validity of our conclusions.

### 6.1 Hypothesis Testing

This study investigates the null hypothesis that modifying Java classes results in similar effectiveness regardless of the use of the Strategy pattern.

Section 4 shows results of evaluating effectiveness by analysing the correctness with which participants performed a task in DAO classes with (experimental treatment) and without (control treatment) the Strategy pattern. Participants performed better for the control treatment, achieving in the control higher correctness rate than in the experimental treatment.

The difference between treatments was found significant at $\alpha = 0.05$, and the effect size was found relevant, according to the literature [16]. Therefore, we **reject** the *null hypothesis* and **accept** the *alternative hypothesis* stating that the correctness to perform a task varies with the use of the Strategy DP.

## 6.2 Research Question

The research question driving our family of experiments asks whether GoF DPs significantly impact maintainability. This is critical given the role of software maintenance in development costs, and the lack of understanding on the impact of DPs on maintainability.

This study does not intend to answer the research question entirely, but it intends to be a one step towards the answer. As any other [39], the experiment reported in this paper constitutes a single piece of evidence rather than a definitive proof.

It would be unrealistic to expect that a single experiment would enable wide generalisations in different contexts [9, 15]. Therefore, independent replications are necessary to increase the confidence in these results and possibly enable their generalization beyond the scope of this study [32].

As Section 6.1 discusses, our results suggest that the Strategy DP may negatively impact software maintainability. In particular, the effectiveness of maintainability. This *does not* mean one should avoid using the DP, but it highlights the importance of a broader evaluation when adopting a particular design practice.

Although this study does not aim to investigate reasons that led to our results, we could observe that implementing the Strategy DP required several additional steps when performing maintenance tasks, compared with the traditional structure of a DAO. These additional steps are proper of the Strategy pattern implementation, as noted by Kerievsky [27].

When assessing tasks, we noted that often these additional steps were not fully completed, leading to incorrect implementation of the task. This could indicate failure in the training process, which could be evaluated by replicating this study with practitioners experienced in the Strategy pattern.

The next step is replicating this experiment with other behavioural patterns in a similar context. This will enable us to understand whether other DPs in the same category will lead to similar results.

# 7 THREATS TO VALIDITY

Although Section 4.1 shows statistical results suggesting that a particular inference is likely to be true with a relative significance, several reasons may threaten our conclusions. This section analyses the main threats to validity we identified in our study.

As identifying these threats is a subjective activity, we used the checklist provided by Shadish et al. [38] as a guideline to identifying possible threats in our study. In addition, we used the threat list provided by Riaz et al. [37] in their literature review of studies investigating software patterns. To analysing the threats, we carried out several follow-up analyses using additional data extracted from our instruments (Section 3.5) and presented in Section 4.

## 7.1 Threats to Statistical Conclusion Validity

These are reasons that may threaten the validity of our conclusions regarding the covariation between treatment and outcome [38].

*7.1.1 Heterogeneity of participants.* Because we used a within-subject design (Section 3.4), our participants took part in both treatments. Therefore, we expect that heterogeneity of participants does not threaten the validity of our conclusion [38]. Nevertheless, we found important to analyse this as the characterisation questionnaire (Section 3.5) revealed heterogeneous participants.

The first characteristic is that seven participants (36.8%) were employed as software developers whereas the remaining 12 participants (63.2%) were not. Because of the practical experience, employed participants could perform better than unemployed participants. However, Section 4.2 shows that the overall result presented in Section 4.1 holds true, regardless of the employment status.

The second characteristic is that ten participants (52.6%) had previous experience with DPs whereas the remaining 9 participants (47.4%) had not. Like employment, the previous experience could benefit a group of participants over another. However, Section 4.3 shows that previous experience with DPs does not significantly change the overall result.

Both employment and previous DP experience among our participants do not differ significantly from results in Section 4.1. Therefore, heterogeneity of participants may not threaten the validity of our conclusions, but they may *moderate* our results.

## 7.2 Threats to Internal Validity

These are alternative explanations for the covariation between treatment and outcome that may confound our conclusions [38].

*7.2.1 Maturation.* Our study may be subject to this threat since participants performed tasks in both treatments. To mitigate this threat, we randomly sorted treatment order (Section 3.4). This resulted into two groups of participants: those who received the experimental treatment first (n = 8), and those who received the experimental treatment latter (n = 11).

However, this strategy does not mitigate the risk of participants' fatigue. Therefore, we analysed correctness regarding the treatment order in which each task was performed to understand whether fatigue represents a threat to our conclusion. Fatigue threatens our conclusion if correctness decreases significantly with time.

Section 4.4 shows that the likelihood of succeeding in the maintenance task increases with time for the Strategy pattern, but decreases with time for the traditional implementation. Therefore, we evaluate that the contradictory data does not provide sufficient evidence that maturation plays a significant role in our conclusion.

*7.2.2 Testing.* As maturation, testing may threaten our conclusion since participants performed tasks in both treatments and therefore, they may have mastered a particular task/treatment over another. We can analyse effects of testing in our results in two ways. First, by analysing the order with which tasks were performed. As presented in Section 7.2.1, data does not suggest that testing plays a role in our conclusion for the same reason it does not play a role in fatigue.

The second way to analyse effects of testing is by comparing correctness measured in the pretest with that measured in the post-test. Testing threatens our conclusion if correctness in the post-test is significantly higher than the pretest. As Section 4.5 shows, the correctness increase observed in our pretest/post-test analyses represent a real threat to our conclusion.

## 7.3 Threats to Construct Validity

These are reasons that may threaten our conclusions regarding instances of constructs used to operationalise our experiment [38].

*7.3.1 Reactivity to the experimental situation.* This threat refers to participants' perceptions on treatments. To investigate whether reactivity to the experimental situation threatens our results, we

could analyse the correlation between participants' results and their perceptions as identified by the feedback form (Section 4.6). However, because the feedback form was filled only after the experiment (Section 3.7), this correlation could not clarify whether their perceptions affected their results or the other way around.

As a conclusion, we argue that reactivity to the experimental situation does not represent a threat since most participants present a positive perspective towards the experimental treatment although they clearly consider the control treatment as the easiest. In addition, we could not find a significant pattern among participants that could suggest a trend towards different results.

## 7.4 Threats to External Validity

Shadish et al. [38] explain that threats to external validity are reasons that may make causal relationships not hold over variations in participants, settings, treatments, and outcomes that were or were not in the study. Whereas Section 7.4.1 discusses whether our conclusion hold over variations of elements that were in our study, Section 7.4.2 discusses whether our conclusion hold over variations of elements that were not in our study.

*7.4.1 Variations of Elements in our Study. Participants.* The variation in participants may threaten our conclusion if correctness vary significantly with participants' characteristics. As we analyse in Sections 7.1.1 and 7.3, it is unlikely that variations in participants threaten the external validity.

*Outcome.* Other possible threat is the outcome variation, which could threaten our conclusion if another outcome variable suggested a different conclusion. In addition to correctness, we also collected participants' perception on the use of the Strategy DP. This could also be analysed as a secondary outcome since participants' perceptions are commonly used as a measure in related studies [1, 37, 48].

As discussed in Section 7.3, participants perceived the control treatment as the easiest to modify although they recognised that the Strategy DP could bring benefits to DAO classes. This result is in line with our results on correctness, since we understand that something easier to modify would result in less errors.

*7.4.2 Variations of Elements not in our Study.* Since our study plan takes several design decisions, our conclusions are mainly limited to the type of participants, settings, treatments, and outcomes we used to plan and conduct our study [15].

However, using principles of generalised causal inference [38], our conclusions may hold over several variations of elements that were not part of our study. First, since the Strategy DP shares similar structure with State and Bridge patterns [21], we could suggest that our results are also valid for these patterns.

Second, our results could be generalised to software developers if our participants share characteristics with software developers. However, there is no widely accepted source that could be used to identify characteristics of a software developer [8]. Therefore, as Baltes & Diehl suggest [8], we analyse our participants against results of the StackOverflow developer survey[7] to check to what extent our participants represent software developers.

Regarding age, our participants can be grouped into two categories. Whereas most participants (73.7%) are between 20 and 24, the five remaining participants (26.3%) are between 25 and 31. The young age is expected since our participants are undergraduates.

In the StackOverflow survey, participants within the same range of ours (20–31) represent nearly 50% of respondents. However, in the StackOverflow survey, participants between 20 and 24 years old represent the second most common category whereas participants between 25 and 29 years old represent the first most common category. In addition, our participants are significantly younger (23.5 years old) than Brazilian participants in the StackOverflow survey (29.4 years old).

Regarding gender, our participants differ significantly from professional developers in the StackOverflow survey. Although most of our participants are male (78.9%) like in the survey (91.7%), our female participants (21.1%) are far more representative than female developers in the StackOverflow survey (7.7%).

As our participants are undergraduates, we checked the Brazilian higher education census[8]. It shows that our participants also differ from common gender distribution in Brazilian higher education.

Not surprisingly, most of our participants (63.1%) are currently unemployed unlike the StackOverflow survey, whereby unemployment among professional developers is about 2.1%. In addition, the StackOverflow survey shows that only 2.1% of professional developers are currently students.

Finally, regarding coding experience, most of our participants (84.2%) have between 2 and 5 years of coding experience whereas in the StackOverflow survey, only 10.5% of professional developers have less than 5 years.

As our participants significantly differ from software developers in the StackOverflow survey for several indicators, we do not advocate generalising our conclusions beyond our participants.

## 8 CONCLUSION

The quasi-experiment reported in this paper shows evidence that the Strategy pattern does impact effectiveness of modifying DAO classes ($p$-value = 0.003, relative risk of failing = 5.5, odds ratio = 10.8). Although this suggests that the Strategy DP reduce maintainability, we highlight that our findings are indicative and we do not claim for generalisation beyond the context of this study.

This study extends the current state-of-the-art by yielding the first evidence on the negative impact of the Strategy DP on maintainability and by carrying out the first study that uses experimentation to investigating the Strategy DP. In addition, we provide a complete package available online for replications and auditing.

Our results contribute to changing the current understanding on the impact of the Strategy DP on maintainability and to increasing the number of studies, which in turn, makes it possible to use aggregation methods to devise a stronger evidence.

## ACKNOWLEDGMENTS

The authors are grateful to all students that took part in this study, to Universidade Tecnológica Federal do Paraná that supported our study, and to Dr Willian Massami Watanabe and Dr Elias Adriano Nogueira da Silva for their valuable feedback on an earlier version of the manuscript.

[7] https://insights.stackoverflow.com/survey/2020/

[8] http://portal.inep.gov.br/web/guest/resumos-tecnicos1

# REFERENCES


[1] Mawal Ali and Mahmoud O. Elish. 2013. A Comparative Literature Survey of Design Patterns Impact on Software Quality. In *2013 International Conference on Information Science and Applications (ICISA)*. IEEE, Suwon, South Korea, 1–7.

[2] Deepak Alur, Dan Malks, and John Crupi. 2003. *Core J2EE Patterns: Best Practices and Design Strategies* (2 ed.). Prentice Hall. 528 pages.

[3] Apostolos Ampatzoglou, Sofia Charalampidou, and Ioannis Stamelos. 2013. Research state of the art on GoF design patterns: A mapping study. *Journal of Systems and Software* 86, 7 (jul 2013), 1945–1964.

[4] Apostolos Ampatzoglou and Alexander Chatzigeorgiou. 2007. Evaluation of object-oriented design patterns in game development. *Information and Software Technology* 49, 5 (may 2007), 445–454.

[5] Frank M. Andrews, Laura Klem, Terrence N. Davidson, Patrick M. O'Malley, and Willard L. Rodgers. 1981. *A Guide for selecting statistical techniques for analyzing social science data* (2nd ed.). Survey Research Center, Institute for Social Research, University of Michigan, Michigan. 71 pages.

[6] Erik Arisholm and Dag I.K. Sjøberg. 2004. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Transactions on Software Engineering* 30, 8 (aug 2004), 521–534.

[7] B. Bafandeh Mayvan, A. Rasoolzadegan, and Z. Ghavidel Yazdi. 2017. The state of the art on design patterns: A systematic mapping of the literature. *Journal of Systems and Software* 125 (mar 2017), 93–118.

[8] Sebastian Baltes and Stephan Diehl. 2016. Worse Than Spam. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '16*. ACM Press, New York, New York, USA, 1–6.

[9] V.R. Basili, F. Shull, and F. Lanubile. 1999. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering* 25, 4 (1999), 456–473.

[10] Andrew Binstock. 2018. Useful Patterns and Best Practices. *Java magazine* (2018), 14–14. https://www.oracle.com/technetwork/java/javamagazine/index.html

[11] J Breaugh. 2003. Effect Size Estimation: Factors to Consider and Mistakes to Avoid. *Journal of Management* 29, 1 (jan 2003), 79–97.

[12] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, New York, NY, USA. 476 pages.

[13] Jeffrey C. Carver, Letizia Jaccheri, Sandro Morasca, and Forrest Shull. 2010. A checklist for integrating student empirical studies with research and teaching goals. *Empirical Software Engineering* 15, 1 (feb 2010), 35–59.

[14] Tore Dybå, Vigdis By Kampenes, and Dag I.K. Sjøberg. 2006. A systematic review of statistical power in software engineering experiments. *Information and Software Technology* 48, 8 (aug 2006), 745–755.

[15] Tore Dybå, Dag I.K. Sjøberg, and Daniela S. Cruzes. 2012. What works for whom, where, when, and why?. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12*. ACM Press, 19.

[16] Paul D. Ellis. 2010. *The Essential Guide to Effect Sizes* (1 ed.). Cambridge University Press, Cambridge, UK. 192 pages.

[17] Janet Feigenspan, Christian Kastner, Jorg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, Passau, 73–82.

[18] A Field, J Miles, and Z Field. 2012. *Discovering Statistics Using R*. SAGE Publications, London, UK. 992 pages.

[19] Eric Freeman, Bert Bates, Kathy Sierra, and Elisabeth Robson. 2004. *Head First Design Patterns: A Brain-Friendly Guide*. O'Reilly Media, Sebastopol. 694 pages.

[20] Andrei Furda, Colin Fidge, Olaf Zimmermann, Wayne Kelly, and Alistair Barros. 2018. Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency. *IEEE Software* 35, 3 (may 2018), 63–72.

[21] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Boston. 395 pages.

[22] Gregor Hohpe and Bobby Woolf. 2003. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, Boston, MA. 736 pages.

[23] Martin Höst, Björn Regnell, and Claes Wohlin. 2000. Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering* 5, 3 (2000), 201–214.

[24] Ronald Jabangwe, Jürgen Börstler, Darja Šmite, and Claes Wohlin. 2014. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering* 20, 3 (mar 2014), 640–693.

[25] Andreas Jedlitschka, Marcus Ciolkowski, and Dietmar Pfahl. 2008. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, Forrest Shull, Janice Singer, and Dag I.K Sjøberg (Eds.). Springer-Verlag, London, UK, Chapter 8, 201–228.

[26] Vigdis By Kampenes, Tore Dybå, Jo E. Hannay, and Dag I. Dag. 2009. A systematic review of quasi-experiments in software engineering. *Information and Software Technology* 51, 1 (2009), 71–82.

[27] Joshua Kerievsky. 2004. *Refactoring to Patterns*. Addison-Wesley Professional. 400 pages.

[28] Foutse Khomh and Yann-Gael Gueheneuce. 2008. Do Design Patterns Impact Software Quality Positively?. In *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE, Athens, 274–278.

[29] Barbara Kitchenham, Lech Madeyski, and Pearl Brereton. 2019. Problems with Statistical Practice in Human-Centric Software Engineering Experiments. In *Proceedings of the Evaluation and Assessment on Software Engineering - EASE '19*. ACM Press, New York, New York, USA, 134–143.

[30] Barbara A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 28, 8 (aug 2002), 721–734.

[31] James Miller. 2004. Statistical significance testing––a panacea for software technology experiments? *Journal of Systems and Software* 73, 2 (oct 2004), 183–192.

[32] James Miller. 2005. Replicating software engineering experiments: a poisoned chalice or the Holy Grail. *Information and Software Technology* 47, 4 (mar 2005), 233–244.

[33] James Miller, John Daly, Murray Wood, Marc Roper, and Andrew Brooks. 1997. Statistical power and its subcomponents — missing and misunderstood concepts in empirical software engineering research. *Information and Software Technology* 39, 4 (apr 1997), 285–295.

[34] Sofia Ouhbi, Ali Idri, Jose Luis Fernandez Aleman, and Ambrosio Toval. 2014. Evaluating Software Product Quality: A Systematic Mapping Study. In *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*. IEEE, Rotterdam, 141–151.

[35] Lesley M. Pickard, Barbara A. Kitchenham, and Peter W. Jones. 1998. Combining empirical results in software engineering. *Information and Software Technology* 40, 14 (dec 1998), 811–821.

[36] Hridesh Rajan, Steven M. Kautz, and Wayne Rowcliffe. 2010. Concurrency by modularity: design patterns, a case in point. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications - OOPSLA '10*. ACM Press, New York, New York, USA, 790.

[37] Maria Riaz, Travis Breaux, and Laurie Williams. 2015. How have we evaluated software pattern application? A systematic mapping study of research design practices. *Information and Software Technology* 65 (sep 2015), 14–38.

[38] W R Shadish, T D Cook, and D T Campbell. 2002. *Experimental and Quasi-experimental Designs for Generalized Causal Inference*. Wadsworth Cengage Learning, Belmont, USA. 656 pages.

[39] Forrest Shull and Raimund L. Feldmann. 2008. Building Theories from Multiple Evidence Sources. In *Guide to Advanced Empirical Software Engineering* (1 ed.), Forrest Shull, Janice Singer, and Dag I. K. Sjøberg (Eds.). Springer London, London, Chapter 13, 337–364.

[40] Dag I.K. Sjøberg, Tore Dybå, and Magne Jorgensen. 2007. The Future of Empirical Methods in Software Engineering Research. In *Future of Software Engineering (FOSE '07)*. IEEE, Minneapolis, MN, 358–378.

[41] Dag I.K. Sjøberg, J.E. Hannay, O. Hansen, Vigdis By Kampenes, A. Karahasanovic, N.-K. Liborg, and A.C. Rekdal. 2005. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering* 31, 9 (sep 2005), 733–753.

[42] Ian Sommerville. 2016. *Software Engineering* (10 ed.). Pearson, Essex. 810 pages.

[43] JAMES SUGRUE. 2017. *DZONE'S GUIDE TO JAVA DEVELOPMENT AND EVOLUTION*. Technical Report. DZone. 46 pages. https://dzone.com/guides/java-development-and-evolution

[44] Sira Vegas, Patricia Riofrio, and Natalia Juristo. 2017. Does Subject Type Influence Software Engineering Experiment Results?. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, Buenos Aires, Argentina, 210–212.

[45] Vaughn Vernon. 2013. *Implementing Domain-Driven Design*. Addison-Wesley Professional. 656 pages.

[46] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. 249 pages.

[47] S.W.L. Yip and T. Lam. 1994. A software maintenance survey. In *Proceedings of 1st Asia-Pacific Software Engineering Conference*. IEEE Comput. Soc. Press, Tokyo, Japan, 70–79.

[48] Cheng Zhang and David Budgen. 2012. What Do We Know about the Effectiveness of Software Design Patterns? *IEEE Transactions on Software Engineering* 38, 5 (sep 2012), 1213–1231.

[49] Cheng Zhang and David Budgen. 2013. A survey of experienced user perceptions about software design patterns. *Information and Software Technology* 55, 5 (may 2013), 822–835.