Impact of the Visitor Pattern on Program Comprehension and Maintenance

Sébastien Jeanmart[1], Yann-Gaël Guéhéneuc[2],
Houari Sahraoui[3], and Naji Habra[1]
[1] PReCISE Research Center, University of Namur, Namur, Belgium
[2] Ptidej Team, DGIGL, École Polytechnique de Montréal, Quebec, Canada
[3] GEODES, DIRO, University of Montreal, Quebec, Canada

E-mails: sjeanmar@student.fundp.ac.be, yann-gael.gueheneuc@polymtl.ca, sahraouh@iro.umontreal.ca, and nha@info.fundp.ac.be

Abstract

In the software engineering literature, many works claim that the use of design patterns improves the comprehensibility of programs and, more generally, their maintainability. Yet, little work attempted to study the impact of design patterns on the developers' tasks of program comprehension and modification. We design and perform an experiment to collect data on the impact of the Visitor pattern on comprehension and modification tasks with class diagrams. We use an eye-tracker to register saccades and fixations, the latter representing the focus of the developers' attention. Collected data show that the Visitor pattern plays a role in maintenance tasks: class diagrams with its canonical representation requires less efforts from developers.

1. Introduction

Much work is based on the *premises* that patterns improve the analysability and changeability of programs, for example [1, 19]. Patterns are the focus of many works studying their relevance, visualisation, identification, and so on, (e.g., [1, 13, 14, 15, 16]), with the hypothesis that their use improves quality.

Yet, some studies suggested that patterns may impact negatively development and maintenance activities because of the complexity that they introduce in the design and implementation of programs and because of the assumption that they are hard to understand by developers. For example, Wendorff [25] assessed the impact of patterns in some industrial projects and showed that they impacted negatively comprehensibility. Bieman *et al.* [4] showed that classes playing certain roles in some design patterns are more change prone that others classes. Khomh and Guéhéneuc [21] performed an empirical study of the impact of the 23 design patterns from [10] on ten different quality characteristics and concluded that patterns do not necessarily promote reusability, expandability, and understandability, as advocated by Gamma *et al.*

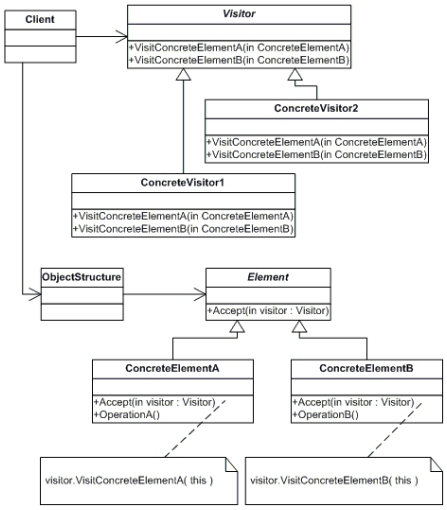


Figure 1. Solution Suggested by the Visitor Design Pattern (canonical representation)

The goal of this study is to determine whether the Visitor pattern is useful for comprehension and maintenance. With the Visitor design pattern, a visitor "[r]epresent[s] an operation to be performed on the ele-

ments of an object structure." The "Visitor lets you define a new operation without changing the classes of the elements on which it operates." [10]. Figure 1 shows the typical solution suggested by the Visitor pattern, in its canonical representation. We study the Visitor pattern because this pattern is widely used in practice and has several design alternatives. Using more than one pattern would have made it difficult to isolate their respective impact on the developers' efforts.

We study the impact of the Visitor pattern on tasks representative of those characteristics: *comprehension* and *modification* tasks and we propose to compare developers' efforts (1) in the presence or not of the Visitor when performing comprehension and modification tasks and (2) with different layouts of the Visitor pattern, for example the one shown in Figure 1 from [10].

We design experiments to collect data to compare the developers' efforts when performing comprehension and modification tasks using different yet *semantically equivalent* UML class diagrams. We define effort as function of the *amount of attention* that developers must spend to perform the tasks: less attention and less time means less effort. We capture the developers' attention using data collected with an eye-tracker to decide whether a class diagram decrease their efforts with respect to the others. We chose UML class diagrams because UML is a *de-facto* standard and its main constructs are well known of developers.

We collected data for three programs and 24 developers. For comprehension tasks, we report that no significant difference exits between class diagrams with, without, or with a modified representation of the Visitor. For modification tasks, we show that, developers perform with significantly less efforts on diagrams where the Visitor is represented as shown in Figure 1 [10]. Additionally, we report that the levels of knowledge of UML and of design patterns do not impact significantly our results.

The remainder of this paper is organised as follows. First, Section 2 presents some related works. The experimental design is described in Section 3. Section 4 presents the running of our experiments, the collected data, and results. We discuss our study and assess its validity in Section 5 and conclude in Section 6.

2. Related Works

This paper relates to previous works on the impact of design patterns on software quality, in particular for evolution and maintenance, program comprehension techniques and models, and eye-tracking. We build on these works to further study the usefulness of the Visitor in details.

Impact of Design Patterns. Vokac *et al.* [22] analysed the corrective maintenance of a large commercial system over three years and compared the defect rates of classes that participated in design patterns against those of classes that did not. They noticed that participating classes were less defect prone than others.

Khomh and Guéhéneuc [21] performed an empirical study of the impact of the 23 design patterns from [10] on ten different quality characteristics and found that these patterns do not always improve the quality of programs. They reported for each quality characteristic the list of patterns that decreases it and concluded that patterns do not necessarily promote reusability, expandability, and understandability.

Other related works include, for example, the studies by Di Penta *et al.* [8] and Bieman *et al.* [4] on the evolution of classes playing roles in design motifs that showed that some role are more change-prone than others; the work by Wydaeghe *et al.* [27] on the concrete use of six design patterns to build an OMT editor and their impact on reusability, modularity, flexibility, and understandability; Wendorff's evaluation [25] of design patterns in a large commercial systems that shows their negative impact.

Program Comprehension. Diagram reading is an important activity in software engineering because all developers use diagrams (or sketches thereof) to convey information to their peers and understand a program.

Purchase performed several studies to provide (1) a better understanding on the use of diagrams by developers (such as entity–relationship diagrams [17]), (2) criteria to assess the aesthetics of diagrams (in particular node–link diagrams [24]), and (3) set of user preferences for graph layouts (specifically in the context of the UML notations [7]).

Several authors proposed and evaluated models of program comprehension. One of the first models, proposed by Brooks [6], described program comprehension as a process of building a sequence of knowledge domains, bridging the gap between problem domain and program execution. Soloway [20] suggested that developers use plans when comprehending a program. Another model by von Mayrhauser [23] described the program comprehension process as a combination of top-down and bottom-up processes, working on a common knowledge base.

Eye-tracking. Eye tracking is mainly a means to analyse the activity of *comprehension*. The fundamental design of eye-trackers is based on human visual capability [9]. The data collected by eye-trackers provides insight on subjects' focus of attention to infer their underlying cognitive processes [18].

For example, Bednarik *et al.* [3] proposed an approach to investigate trends in repeated-measures sparse-data of few cases captured by an eye-tracker. Using this approach, they characterised the development of comprehension strategies during program visualisation. They also studies program debugging [2].

Independently to Bednarik *et al.*'s study, the second author of this study conducted an experiment to understand the use of information from UML-like class diagrams by developers [12]. He concluded that classes and interfaces are important and on that binary class relationships are little used.

A similar study has been carried out by Yusuf *et al.* [28] to identify the relative importance of specific characteristics of UML-like class diagrams during comprehension, such as layout, color, and stereotypes. They also studied how developers navigate in class diagrams. They concluded that layouts with additional information were the most effective and that the use of stereotypes plays an important role during comprehension.

We build on this previous work to develop our studies and to use an eye-tracker to collect relevant data.

3. Experiments Design

The design of our experiments is directed by our use of an eye-tracker to collect relevant data. All data is available on-line for replication[1].

3.1. Eye-trackers

Many eye-trackers exist on the market. We use the EyeLink II from SR Research[2]. EyeLink II has the highest resolution (noise-limited at $< 0.01°$) and fastest data rate (500 samples per second) of any head mounted eye-tracker. Such an eye-tracker is composed of two computers and a head-band. The headband includes two cameras and an infra-red emitter. The cameras use infra-red rays that are reflected on the subject's cornea to register eye-movements. Four sensors are placed on the subject's screen. These sensors work with the infra-red emitter to gather the position of the head-band with respect to the screen. With the head-band cameras and these four sensors, the system computes the position of the subject's gaze on the screen.

The data collected from the eye-trackers are of two types: raw positions and parsed positions. Parsed positions are expressed in terms of fixations and saccades based on physiological thresholds. Fixations are stabilisations of the eye during a gaze while saccades are movements from one fixation to another. The numbers of fixations and saccades related to a given task can vary from a dozen to several hundreds depending on the size and complexity of the class diagram. These two pieces of data can be combined to measure the subjects' efforts.

[1] `http://www.ptidej.net/downloads/experiments/esem09/`
[2] `http://www.eyelinkinfo.com/`

3.2. Hypotheses

We want to assess the following four null hypotheses, two for comprehension tasks (HC) and two for modification tasks (HM):

- HC_{0_1}: A class diagram with the Visitor does not reduce the subjects' efforts during program comprehension when compared to a class diagram without it.
- HC_{0_2}: A class diagram using the canonical representation of the Visitor does not reduce the subjects' efforts during program comprehension when compared to a class diagram using the Visitor with another layout.
- HM_{0_1}: A class diagram with the Visitor does not reduce the subjects' efforts during program modification when compared to a class diagram without.
- HM_{0_2}: A class diagram using the canonical representation of the Visitor does not reduce the subjects' efforts during program modification when compared to a class diagram using the Visitor with another layout.

If the previous null hypotheses are rejected, we will assess whether alternative hypotheses are supported:

- HC_{a_1}: A class diagram with the Visitor reduces the subjects' efforts during program comprehension when compared to one without it.
- HC_{a_2}: A class diagram using the canonical representation of the Visitor reduces the subjects' efforts during program comprehension when compared to one using the Visitor and another layout.
- HM_{a_1}: A class diagram with the Visitor reduces the subjects' efforts during program modification when compared to a class diagram without it.
- HM_{a_2}: A class diagram using the canonical representation of the Visitor reduces the subjects' efforts during program modification when compared to one using the Visitor and another layout.

3.3. Independent Variables

From the previous hypotheses, we identify the following independent variables:

- DESIGN ALTERNATIVE: CP, MP, and NP are the possible values for this variable. Each class diagram conveys the same semantics and has a similar complexity. NP diagrams (*no pattern*) do not use the pattern; CP diagrams (*canonical pattern*) show the canonical representation of the pattern, for example see Figures 1; MP diagrams (*modified pattern*) show the pattern with a modified layout. (No systematic layout modification has been performed because we only want to assess the familiar layout against a unfamiliar layout.)
- TASKS: Two different tasks are compared, one program comprehension task and one modification task: C or M.

In the rest of this paper, the different experiments performed by the subjects are named as: "DESIGN_ALTERNATIVE"_"TASK", for example CP_C stands for a diagram with the *Canonical Pattern* used in a *Comprehension* task.

3.4. Mitigating Variables

We retain two mitigating variables to put into perspective the results of our experiments:

- UML KNOWLEDGE: The subjects' knowledge of UML, established using a questionnaire of our own design. Value range is $[1, 2]$ where 2 means that a subject has a very good knowledge of UML and 1 that the subject knows UML.
- DP KNOWLEDGE: The subjects' knowledge of design patterns. This knowledge is also established using a questionnaire and uses the same value range as UML KNOWLEDGE.

3.5. Dependent Variables

We use fixations to measure a subject's *attention* in an area of the screen. We create for each diagram (NP, CP, and MP) two lists of *areas of interest* and *areas of relevant interest*; these include, respectively, *all classes* and *all relevant classes* to perform a task.

An area of interest is any class in our diagrams that could be the focus of the subjects' attention to perform comprehension or modification tasks (C or M) defined in Section 3.7. An area of relevant interest is any class that is *relevant* for the task at hand and, therefore, should receive *more* subjects' attention during the task.

We use the following three dependent variables:

1. **ADRF and ADNRF.**

   We conjecture that, in a first phase subjects look at the different classes of the diagram to *identify* relevant ones. Then, in a second phase, they use mainly relevant classes to *perform* the task. The difference in time due to the Visitor, if it exists, should be in the second phase.

   The variable ADRF denotes the Average Duration of Relevant Fixations and ADNRF denotes the Average Duration of Non-Relevant Fixations. They are defined as follows:

   $$ADRF = \frac{\sum_{c \in \{Rel.\ Classes\}} d(c)}{\#\{Rel.\ Classes\}}$$

   $$ADNRF = \frac{\sum_{c \in \{NonRel.\ Classes\}} d(c)}{\#\{NonRel.\ Classes\}}$$

   where $d(c)$ is a function that gives the total *duration* of the fixations made to a class c.

   Different design alternatives (CP, MP, and NP) may have different number of classes. To avoid favoring diagrams with fewer classes, we take the average time per class rather than the total time.

   Following our conjecture, ADRF should differ between alternatives, ADNRF should be constant.

2. **NRRF.**

   We conjecture that if a diagram requires more *relevant fixations* than another (normalised by the overall number of fixations to account for diagrams with more or less classes), then the corresponding representation requires more of the subjects' efforts to perform a same task and, therefore, it is less efficient, (a *higher* rate means *more* efforts).

   The variable NRRF, Normalised Rate of Relevant Fixations [11], measures the subjects' efforts when performing tasks on diagrams, it is defined as:

   $$NRRF = \frac{\frac{\sum_{c \in \{Rel.\ Classes\}} f(c)}{\#\{Rel.\ Classes\}}}{\frac{\sum_{c \in \{Rel.\ Classes\} \cup \{NonRel.\ Classes\}} f(c)}{\#\{Rel.\ Classes\} + \#\{NonRel.\ Classes\}}}$$

   where $f(c)$ is a function that, given a class c, returns the number of *fixations* performed by a subject in that class. We again normalise the number of relevant fixations (numerator) by the numbers of fixations in relevant and irrelevant classes.

We also use the answers given by the subjects aloud and recorded using a pre-defined questionnaire to assess their correctness systematically. All subjects answered correctly but with variations in fixations and time. Thus, in the following, we only study the dependent variables ADRF, ADNRF, and NRRF.

3.6. Objects and Subjects

We choose three open-source programs as objects: JHotDraw, JRefactory, and PADL. JHotDraw is a framework to implement technical and structured drawings. This framework provides support for the creation of geometric and user-defined shapes. JRefactory is a code refactoring tool for Java programs. PADL is a meta-model for describing patterns and object-oriented programs. These programs all use the Visitor pattern. Since full documentation was not available, we obtain their UML-like class diagrams through partial reverse engineering. For each of these three programs, three semantically-equivalent versions of their structure have been designed.

The experiment was performed with 24 subjects: 7 post-graduate and 17 graduate students at the Department of Informatics and Operations Research at University of Montreal. All students have designed software architectures and used UML class diagrams. These 24 subjects are randomly placed into 3 balanced groups. The balancing simplifies and strengthen our statistical analysis of the collected data [26].

3.7. Questions

Appropriate questions must trigger the subjects' mental processes when performing their tasks [3]. We ask specific questions to the subjects with each design alternatives. Half the questions focuses on the comprehension of the diagram, the other half on a modification task. Questions were designed in such a way that each class diagram contained enough information to answer the questions. Typically, questions involved two or three classes and took no more than seven minutes to answer comprehension/modification tasks.

4. Experiments and Analyses

4.1. Experiments

The experiments were approved by the ethical review board (ERC) of University of Montreal. Upon arrival of a subject, we reminded her the general goal of the study, stated as "To analysis your behaviour when performing comprehension and maintenance tasks with UML class diagrams". No mention of design patterns was made to avoid any possible bias. We also reminded her that she could leave the experiment at any time without any penalty.

Then, we gave a tutorial to the subject on the different constructions typically found in UML class diagrams to level possible individual differences in knowledge. We then simulated a real but smaller experiment to familiarise her with the questions they were to answer. Finally, we gave a short technical explanation about the eye-tracker used to collect data.

Then, the eye-tracker was set and calibrated. Before showing a diagram, a text was displayed with the purpose and main characteristics of the program, enough information to perform the tasks, as confirmed in the post-mortem questionnaire.

Each subject performed six tasks by answering three *comprehension* and three *modification* questions on two different diagrams. The questions were randomised to prevent any learning bias while ensuring that each question was answered for each diagram by an equal number of subjects. No time limits were set but subjects were asked to answer as soon as they thought that they knew the answer.

Finally, we provided each subject with a short post-mortem questionnaire to evaluate their knowledge of UML and of design patterns. We provided this questionnaire after the experiment not to reveal the purpose of the experiment.

The experiments were conducted in a quiet disturbance-free room. For each subject, an experiment took about one hour, including a tutorial (15–20 minutes), data collection (20–30 minutes), and questionnaire (5 minutes).

4.2. Analyses of Comprehension Task Data

The samples measured by the independent variables are independent because they come from unique subjects performing independent tasks. To determine the statistical significance of the observed differences among dependent variables, we use the t-test for normally distributed variables and the Mann-Whitney test for the others, using the significance level $\alpha = 0.05$. All the variables are normally distributed except for ADNRF for comprehension tasks. We attempt to reject the null-hypotheses in favor of the alternative hypotheses. We also explore the two mitigating variables, UML Knowledge and DP Knowledge.

4.2.1 ADRF and ADNRF

Table 1 summarises the ADRF and ADNRF of design alternatives for comprehension tasks. We ob-

(ms)	CP_C	MP_C	NP_C	$\Delta 1$ (NP_C, C_PC)	$\Delta 2$ (NP_C, MP_C)	$sig.$ $(\Delta 1)$	$sig.$ $(\Delta 2)$
ADRF	12,766	12,252	13,332	4%	8%	0.807	0.592
ADNRF	3,355	3,578	3,388	1%	-6%	0.711	0.635
(%)	CP_C	MP_C	NP_C	p-value (HC_{01})		p-value (HC_{02})	
NRRF	75.77	77.95	81.49	0.221		0.663	

Table 1. Effect of Visitor on Comprehension

(ms)	CP_M	MP_M	NP_M	$\Delta 1$ (NP_M, CP_M)	$\Delta 2$ (NP_M, MP_M)	$sig.$ $(\Delta 1)$	$sig.$ $(\Delta 2)$
ADRF	12,138	13,175	18,724	35%	30%	0.045	0.076
ADNRF	4,804	4,024	4,421	-9%	9%	0.672	0.626
(%)	CP_M	MP_M	NP_M	p-value (HM_{01})		p-value (HM_{02})	
NRRF	71.97	73.46	80.18	0.027		0.725	

Table 2. Effect of Visitor on Maintenance

serve that subjects spent 78% of their time on relevant classes for all the alternatives. For example, ADRF was $12,766$ ms for CP_C when ADNRF was $3,355$ ms: after a short time, subjects identified relevant classes.

Regarding hypothesis HC_{0_1}, the time spent looking at non-relevant classes is almost the same for CP_C and NP_C (an improvement of 1% when the Visitor is present). The improvement in time for ADRF is slightly better (4%). However, both improvements are not statistically significant. Thus, we cannot reject the null hypothesis HC_{0_1} and conclude that the Visitor does not reduce significantly the effort when performing comprehension tasks.

We make the same observations when the Visitor is present with a different layout (MP_C vs. NP_C). Indeed, differences between alternatives are small (-6% for ADNRF and 8% for ADRF) and statistically non significant. Therefore, we cannot reject HC_{0_2} either.

4.2.2 NRRF

Table 1 summarises the NRRF for comprehension tasks. NRRF is given respectively for the three design alternatives CP_C, MP_C, and NP_C. The p-values report the statistical significance of the differences between CP_C and NP_C (HC_{0_1}) and MP_C and NP_C (HC_{0_2}).

We observe similar NRRF for CP_C and MP_C (75.77% and 77.95%). The NRRF for NP_C is higher (81.49%) than for CP_C and MP_C. In conclusion, the p-values presented in Table 1 for NRRF indicate that the presence of the Visitor as well as its layout do not have a significant impact on the subjects' efforts. We cannot reject HC_{0_1} and HC_{0_2}.

Moreover, the distributions presented in Figure 2(a) show that the medians for NRRF are around 75% and 85% for all tasks. However, there is a large variance for NRRF between subjects working on MP_C, *i.e.*, a wider box plot, compared to subjects working on CP_C and NP_C. There is also a wide variance for the first quartile of CP_C. Values range from 38% to 67%. These variances could indicate that the subjects' levels of knowledge in UML and–or design patterns play a role when the Visitor is present in the diagrams.

4.3. Analyses of Modification Task Data

4.3.1 ADRF and ADNRF

Collected maintenance data is summarized in Table 2. The proportion of time dedicated to relevant classes is also important and varies between 70% and 80%.

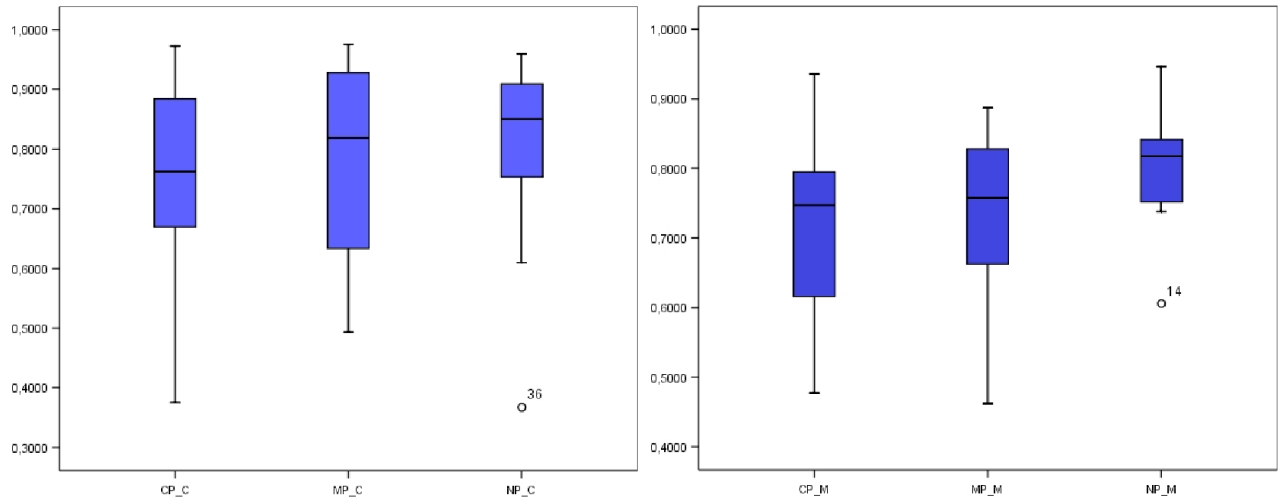
For hypothesis HM_{0_1}, an important reduction in time is observed when the Visitor is present (35% less for CP_M compared to NP_M). This difference is statistically significant (p-value $= 0.045$). We also noticed that subjects used slightly more time on non-relevant classes (-9% for NP_M compared to CP_M). This difference is however not statistically significant (p-value $= 0.672$) and does not compensate for the reduction of ADRF.

We conclude that the null hypothesis HM_{0_1} can be rejected. Consequently, the Visitor does reduce the effort when answering maintenance questions.

For hypothesis HM_{0_2}, we observe a reduction in time of 30% in favor of MP_M compared to NP_M. This reduction is not statistically significant, although the p-value is small (0.076). Thus, we cannot reject the null hypothesis HM_{0_2}.

4.3.2 NRRF

The differences for NRRF are important when comparing tasks with CP_M and MP_M with tasks with NP_M: between 6% and up to 8%, see Table 2.



(a) Comprehension (b) Maintenance

Figure 2. Comprehension and Maintenance Data Distribution (ordinate: NRRF values)

We conclude, using the p-values shown for NRRF in Table 2, that the Visitor with its classical layout has a significant impact on the modification task (p-values $= 0.027$). The impact is not significant with a modified layout (p-values $= 0.725$).

Figure 2(b) shows that subjects working on diagrams with the Visitor pattern and its classical layout have lower values for NRRF. Subjects working on diagrams without the Visitor have a uniform effort, *i.e.*, flat box plot, compared to those of the other groups.

Therefore, we can reject the null hypothesis HM_{0_1} in favor of its alternative hypothesis HM_{a_1}: tasks using CP_M require less effort than using MP_M or NP_M. We cannot reject HM_{0_2}. These results are consistent with the results obtained using ADRF and ADNRF.

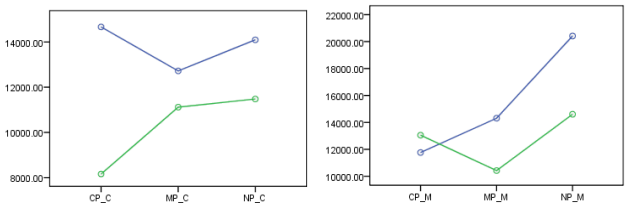
4.4. Impact of Mitigating Variables

We now investigate whether the levels of knowledge in UML and DP mitigate the results.

4.4.1 ADRF and ADNRF

We study the impact of UML Knowledge and DP Knowledge using estimated marginal means because 2-way ANOVA tests were non significant for the individual and combined impacts of the variables.

UML Knowledge. As shown in Figure 3(a), subjects with very good knowledge in UML (green line) perform better for comprehension than others (blue line) when the Visitor is present, *i.e.*, an important difference in the estimated marginal means of ADRF



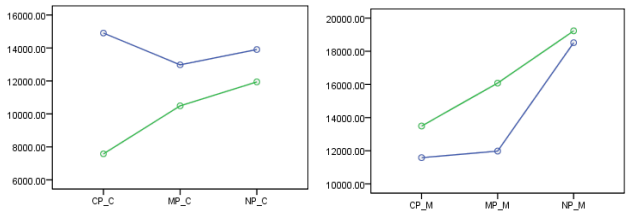
(a) Comprehension (b) Maintenance

Figure 3. Impacts of UML Knowledge and Visitor (ordinate: ADRF estimated mean variances for level 1 (blue) and 2 (green))

between CP_C and NP_C (from roughly $11,500$ ms to $8,000$ ms). There is almost no difference between CP_C and NP_C for subjects with good knowledge.

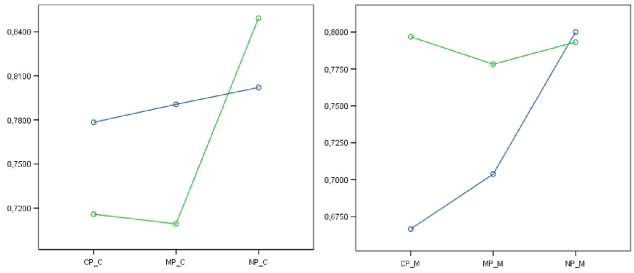
For modification, see Figure 3(b), there is almost no difference between subjects (levels 1 and 2) for CP_M, where the difference is relatively important for NP_M. This observation suggests that the Visitor helps subject with good knowledge in UML, but does not make a real difference for subjects with very good knowledge. It can even be counterproductive for this category of subjects if it appears with a different layout (values for MP_M are less good than the ones for NP_M).

DP Knowledge. The results for comprehension are similar to those of UML, see Figure 4(a): the more the subjects were knowledgeable in design patterns, the more efficient was the Visitor. The impact is minimal for subjects with only good knowledge.



(a) Comprehension (b) Maintenance

Figure 4. Impacts of DP KNOWLEDGE and Visitor (ordinate: ADRF estimated mean variances for level 1 (blue) and 2 (green))



(a) Comprehension (b) Maintenance

Figure 5. Impacts of UML KNOWLEDGE and Visitor (ordinate: NRRF estimated mean variances for level 1 (blue) and 2 (green))

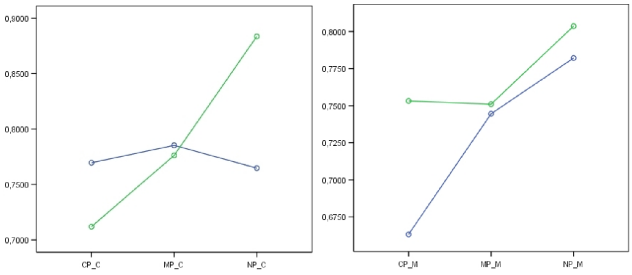
The results obtained for maintenance are mixed, as shown in Figure 4(b). Although there is a significant impact of the presence of the Visitor in maintenance tasks, this impact is attributed to subject with good knowledge, not to those with very good knowledge. More generally, subjects with good knowledge performed better in maintenance tasks than ones with very good knowledge for the three alternatives.

4.4.2 NRRF

paragraphUML KNOWLEDGE Figure 5(a) shows the impact of UML knowledge on comprehension tasks. On NP_C diagrams, subjects who have a very good knowledge of UML (level 2) perform better than those having only a good knowledge (level 1).

For the modification task, in Figure 5(b), subjects with very good UML knowledge (level 2) perform better than other subjects. This result might explain why subjects with better UML knowledge maintain programs more easily.

However, the impact of the UML knowledge cannot be confirmed by a 2-way ANOVA test. Table 3 (a) gives the test significance values for the impact of



(a) Comprehension (b) Maintenance

Figure 6. Impacts of DP KNOWLEDGE and Visitor (ordinate: NRRF estimated mean variances for level 1 (blue) and 2 (green))

(a) UML KNOWLEDGE

NRRF	Comprehension	Maintenance
UML KNOWLEDGE	0.494	0.061
Combined	0.471	0.267

(b) DP KNOWLEDGE

NRRF	Comprehension	Maintenance
DP KNOWLEDGE	0.692	0.245
Combined	0.217	0.546

Table 3. 2-way ANOVA Tests of the Impacts of UML KNOWLEDGE and DP KNOWLEDGE

UML KNOWLEDGE and the combined impact (DESIGN ALTERNATIVE × UML KNOWLEDGE) for comprehension and modification tasks. All p-values are greater than 0.05. Moreover, a p-value of 0.06 indicates that our observation on the impact on modification need to be explored in a replication study.

DP Knowledge. For the impact of DP KNOWLEDGE, we notice an important difference in behavior on NP_C between subjects with a very good knowledge (level 2) and a good knowledge (level 1) of design patterns. The same behavior can be seen for CP_M. Figures 6(a) and 6(b) show differences of roughly 10% for NRRF.

We observe also that for modification tasks, in Figure 6(b), there is a difference between groups of subjects (levels 1 and 2) on all the design alternatives. This difference is an indication that the level of knowledge of design patterns has an impact on modification tasks. Yet, 2-way ANOVA tests give p-values greater than 0.05, as shown in Table 3 (b).

5. Discussions

5.1. Lessons Learned

Although eye-trackers are expansive and have a non-negligible learning curve, they provide useful and help-

ful data to understand the subjects' cognitive processes when performing development and–or maintenance.

The insights gained from their use has led to surprising findings, for example that binary class relationships do not seem to be much of uses to developers [12, 28].

5.2. Threats to the Validity

Internal Validity. We identified three possible threats to internal validity: maturation, instrumentation, and diffusion of the treatments.

In the case of *maturation*, we addressed the learning and fatigue effects by presenting the different tasks (questions to answer) in random and different orders to subjects. We reduced the fatigue effect also by limiting the subjects' effort in time to 20 to 30 minutes.

The *instrumentation* threat is related to the use of the eye-tracker. Subjects had to wear a headband with the infrared camera. They had to minimise their head movements to avoid decalibration. To circumvent this threat, we analysed eye-movement movies recorded during the experiment of each subject. We only detected one case of decalibration and the corresponding data was discarded.

Finally, to prevent subjects from learning the treatments before hand, we gave instructions to each subject to not talk about the experiment before a fixed date corresponding to the end of the experiments with all subjects. We are confident that the instructions were followed by the subjects considering their perceived motivation.

Construct Validity. During the study, we addressed four threats to construct validity : mono-operation bias, mono-method bias, hypothesis guessing, and apprehension. Concerning the two first threats, we used the diagrams extracted from three programs with different application domains. We used three dependent variables, ADRF, ADNRF, and NRRF, to avoid the mono-method bias.

We did not inform the subjects about the goal of the study before hand to anticipate the hypothesis guessing. They were not aware about presence of the patterns in the diagrams. We just explained them in the tutorial that they had to perform tasks on different diagrams coming from different programs. No subject had more than one version of the same diagram.

Finally, different actions were performed to prevent the apprehension threat. First, the eye-tracker was explained to the subjects and they were reassured about the absence of risks related to infrared camera on their eyes. Second, the subject were assigned identifiers to assure that no relation could be traced between them and their data. Finally, although the we ask the subjects to perform diligently, we did not set a predefined time to avoid time pressure.

External Validity. For external validity threats, we considered the interactions of selection and setting with the treatments. The issue of whether student subjects are representative of software professionals has been discussed in several studies (see [5] for example). In our case, we used graduate and postgraduate students that have knowledge of UML and design patterns comparable to most junior professionals. For the setting interaction, we considered the size and the complexity of the diagrams. We reverse-engineered them from open-source programs that are extensively used. The diagrams contained roughly 20 classes, which is usual for comprehension and modification tasks.

Conclusion Validity. Threats to conclusion validity relate to random irrelevancy in setting and heterogeneity of subjects. To prevent the first threat, we used a quiet laboratory. We performed the experiment several times with some subjects (not included in the study sample) to detect and fix in advance any problems. Regarding the choice of subjects, although our sample might be considered as relatively small, the division into three groups combined with the use of three different systems allows deriving 24 observations for each design alternative. This number is generally acceptable for the statistic techniques used. Our students have different backgrounds and different GPAs, thus are heterogeneous. Many of them have work experience. However, we used UML and DP student knowledge as mitigating variables and verified that their impacts are less important than the presence of the pattern.

6. Conclusions and Future Work

The goal of our work was to determine whether the Visitor pattern is useful for maintenance through comprehension and modification tasks. We compared developers' efforts in the presence and absence of the Visitor pattern when performing such tasks and when it is shown as described in [10] and with a different layout.

The analysis of the data collected using an eye-tracker showed that the Visitor pattern does not reduce the subjects' efforts for comprehension tasks. However, the subjects' familiarity with patterns and UML (in general) have observable (even though not significant) influences on comprehension and modification tasks. The analysis showed also that the Visitor pattern with its *canonical* representation reduces the developers' efforts for modification tasks: We could *not* reject HC_{0_1} and HC_{0_2}; we could reject the null hypothesis HM_{0_1}

for its alternative HM_{a_1}, but could *not* reject HM_{0_2}.

The results obtained using three different variables ADRF, ADNRF, and NRRF were consistent. Therefore, we argue that our conjectures hold: subjects need time to identify areas of relevant interests and, when identified, will spend more time studying these areas. However, between two diagrams, more time spent in this areas means a less efficient representation.

In conclusion, the largely-admitted intuition that design patterns may help developers during comprehension and maintenance tasks seemed confirmed on the Visitor pattern only for modification tasks. Therefore, other experiments are required to confirm our results and conclude for other design patterns.

Future work includes investigating design patterns as *communication* artifacts, *i.e.*, variables related to the shape and syntax of diagrams. It also includes replication studies to confirm our observations; to confirm our observations using other variables than ADRF, ADNRF, and NRRF; to confirm our observations using other design patterns. We are also currently conducting studies with experienced professionals to further understand the impact of patterns on their daily activities.

References

[1] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso, and M. Penta. An Empirical Study on the Evolution of Design Patterns. In *ACM SIGSOFT symposium on The Foundations of Software Engineering*, pages 385–394, 2007.

[2] R. Bednarik and M. Tukiainen. Visual attention tracking during program debugging. In *The Third Nordic Conference on Human-Computer Interaction*, pages 331–334, 2004.

[3] R. Bednarik and M. Tukiainen. An eye-tracking methodology for characterizing program comprehension processes. In *2006 symposium on Eye tracking research & applications*, pages 125–132, 2006.

[4] J. Bieman, G. Straw, H. Wang, P. W. Munger, and R. T. Alexander. Design patterns and change proneness: An examination of five evolving systems. In M. Berry and W. Harrison, editors, *Proceedings of the* 9^{th} *international Software Metrics Symposium*, pages 40–49. IEEE Computer Society Press, September 2003.

[5] L. C. Briand, Y. Labiche, M. D. Penta, and H. Yan-Bondoc. An experimental investigation of formality in UML-based development. *Transaction on Software Engineering*, 31(10):833––849, October 2005.

[6] R. Brooks. Using a behavioral theory of program comprehension in software engineering. In M. V. Wilkes, L. Belady, Y. H. Su, H. Hayman, and P. Enslow, editors, *Proceedings of the* 3^{rd} *International Conference on Software Engineering*, pages 196–201. IEEE Computer Society Press, May 1978.

[7] H. C.Purchase, J.-A. Allder, and D. Carrington. Graph layout aesthetics in UML diagrams: User preferences. *journal of Graph Algorithms and Applications*, 6(3):255–279, June 2002.

[8] M. Di Penta, Luigi Cerulo, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study of the relationships between design pattern roles and class change proneness. In H. Mei and K. Wong, editors, *Proceedings of the* 24^{th} *International Conference on Software Maintenance*. IEEE Computer Society Press, September–October 2008.

[9] A. T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag, 2003.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1^{st} edition, 1994.

[11] J. H. Goldberg and X. P. Kotval. Computer interface evaluation using eye movements: Methods and constructs. *International Journal of Industrial Ergonomics*, 24(6):631–645, October 1999.

[12] Y.-G. Guéhéneuc. TAUPE: Towards understanding program comprehension. In H. Erdogmus and E. Stroulia, editors, *Proc. of the* 16^{th} *IBM Centers for Advanced Studies Conference*, pages 1–13. ACM Press, October 2006.

[13] J. K. H. Mak, C. S. T. Choy, and D. P. K. Lun. Precise Modeling of Design Patterns in UML. In *Proceedings of the 26th International Conference on Software Engineering*, pages 252–261, 2004.

[14] T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu. Work experience versus refactoring to design patterns: a controlled experiment. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'06/FSE-14)*, pages 12–22, 2006.

[15] L. Prechelt, B. Unger, M. Philippsen, and W. F. Tichy. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. In *IEEE Transactions on Software Engineering*, volume 28(6), pages 595–606, June 2002.

[16] L. Prechelt, B. Unger, W. F. Tichy, P. Brössler, and L. G. Votta. A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions. In *IEEE Transactions on Software Engineering*, volume 27(12), pages 1134–1144, December 2001.

[17] H. C. Purchase, R. Welland, M. McGill, and L. Colpoys. Comprehension of diagram syntax: An empirical study of entity relationship notations. *International Journal of Human-Computer Studies*, 61(2):187–203, August 2004.

[18] K. Rayner. Eye Movements in Reading and Information Processing: 20 Years of Research. *Psychological Bulletin 124*, pages 372–422, 1998.

[19] A. Shalloway and J. R. Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley, 2000.

[20] E. Soloway, J. Pinto, S. Letovsky, D. Littman, and R. Lampert. Designing documentation to compensate for delocalized plans. *Communication of the ACM*, 31(11):1259–1267, November 1988.

[21] Foutse Khomh and Y.-G. Guéhéneuc. Do design patterns impact software quality positively? In C. Tjortjis and A. Winter, editors, *Proceedings of the* 12^{th} *Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, April 2008. Short Paper.

[22] M. Vokác. Defect frequency and design patterns: An empirical study of industrial code. *Transactions on Software Engineering*, 30(12):904–917, 2004.

[23] A. von Mayrhauser. Program comprehension during software maintenance and evolution. *IEEE Computer*, 28(8):44–55, August 1995.

[24] C. Ware, H. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, June 2002.

[25] P. Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In P. Sousa and J. Ebert, editors, *Proceedings of* 5^{th} *Conference on Software Maintenance and Reengineering*, pages 77–84. IEEE Computer Society Press, March 2001.

[26] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 1^{st} edition, December 1999.

[27] B. Wydaeghe, K. Verschaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers. Building an OMT-editor using design patterns: An experience report. 1998.

[28] S. Yusuf, H. Kagdi, and J. I. Maletic. Assessing the comprehension of UML diagrams via eye tracking. In E. Stroulia and P. Tonella, editors, *Proceedings of the* 15^{th} *IEEE International Conference on Program Comprehension*, pages 145–154. IEEE Computer Society Press, June 2007.