# The Effect of Noise on Software Engineers' Performance

Simone Romano
University of Basilicata
Potenza, Italy
simone.romano@unibas.it

Giuseppe Scanniello
University of Basilicata
Potenza, Italy
giuseppe.scanniello@unibas.it

Davide Fucci
University of Hamburg
Hamburg, Germany
fucci@informatik.uni-hamburg.de

Natalia Juristo
Universidad Politecnica de Madrid
Madrid, Spain
natalia@fi.upm.es

Burak Turhan
Monash University
Melbourne, Australia
burak.turhan@monash.edu

## ABSTRACT

*Background:* Noise, defined as an unwanted sound, is one of the commonest factors that could affect people's performance in their daily work activities. The software engineering research community has marginally investigated the effects of noise on software engineers' performance.

*Aims:* We studied if noise affects software engineers' performance in: *(i)* comprehending functional requirements and *(ii)* fixing faults in source code.

*Method:* We conducted two experiments with final-year undergraduate students in Computer Science. In the first experiment, we asked 55 students to comprehend functional requirements exposing them or not to noise, while in the second experiment 42 students were asked to fix faults in Java code.

*Results:* The participants in the second experiment, when exposed to noise, had significantly worse performance in fixing faults in source code. On the other hand, we did not observe any statistically significant difference in the first experiment.

*Conclusions:* Fixing faults in source code seems to be more vulnerable to noise than comprehending functional requirements.

## CCS CONCEPTS

• **Software and its engineering → Software creation and management**;

## KEYWORDS

Noise, controlled experiment, functional requirement, bug fixing

**ACM Reference Format:**
Simone Romano, Giuseppe Scanniello, Davide Fucci, Natalia Juristo, and Burak Turhan. 2018. The Effect of Noise on Software Engineers' Performance. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '18), October 11–12, 2018, Oulu, Finland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3239235.3240496


*ESEM '18, October 11–12, 2018, Oulu, Finland*

ACM ISBN 978-1-4503-5823-1/18/10…$15.00
https://doi.org/10.1145/3239235.3240496

## 1 INTRODUCTION

Peopleware refers to one of the three aspects of computer technology (hardware and software are the other two). It concerns anything that has to do with the role of people in software development [9]. Peopleware might cover issues related to productivity, organizational factors, workspaces, and so on [2].

Nowadays, workspaces tend to have less privacy, with less dedicated space, which leads to noisy environment. The reason for this trend is the cost. A "penny" saved on the workspace is a "penny" earned on the bottom line, or so the logic goes [13]. The savings of a cost-reduced workplace are attractive, but they need to be compared to the risk of performance reduction in daily working activities/tasks. Software companies that provide a noisy workplace are comforted by the belief that this factor does not matter [13], but noise exerts its specific influences on various forms of cognitive responses [37]. After all, software engineers are knowledge workers—they need to have their brain in gear to do their work—and thus their performance would be sensitive to a noisy workplace.

In this paper, we present the results of two controlled experiments, whose overarching goal was to assess if noise influences software engineers' performance. In the first experiment, we asked 55 final-year undergraduate students in Computer Science to comprehend functional requirements in normal conditions or exposing them to noise. The results indicated the absence of a statistically significant difference in the comprehension of functional requirements. Bearing in mind that noise could exert its influences on people's performance and these influences could be related to the task [6, 18, 37], we asked the students in the first experiment to take part in a second one (42 agreed to take part in), where we varied the kind of software engineering task. That is, participants were asked to fix faults in Java source code. The participants in the second experiment had significantly worse performances in fixing faults in source code when exposed to noise.

The main contribution of our paper is to show the outcomes of the first empirical investigation on the effect of noise in the comprehension of functional requirements and in the fixing of faults in Java code. The results suggest that there are more resource-demanding tasks and noise seems to negatively impact the performances of software engineers when dealing with this kind of tasks.

**Paper Structure.** In Section 2, we present related work and background. We show the design of our investigation in Section 3, while the obtained results are highlighted and discussed in Section 4 and Section 5, respectively. Final remarks conclude the paper.

## 2 RELATED WORK AND BACKGROUND

In this section, we first review the literature related to our study, and then we summarize the reference theories concerning the effects of noise on individuals' performance.

### 2.1 Related Work

The study of *developers' experience*—the considerations that software developers have towards their professional activities—focuses on personal characteristics, such as feelings [17], motivations [14], and workflow [23], while neglecting the physical environment in which they operate. Nevertheless, a better developers' experience is thought to improve not only the job quality of software developers but also their productivity.

There have been a series of studies trying to understand how a particular mental state of software developers impacts their activities. *Flow* is a state of high concentration that results in an absolute assimilation in the activity at hand (*e.g.,* software development) [10]. Disturbance from the surrounding environment in which the activity is taking place (*e.g.,* due to noise) can disrupt a state of flow causing work fragmentation and negative impact on productivity [26]. Alongside, Meyer *et al.* [27] found that developers feel productive when they do not need to switch between tasks and are not interrupted.

Researchers have long recognized the detrimental effects of disturbance in the developers' workplace; subsequently the software engineering and HCI (Human Computer Interaction) communities have proposed different solutions to the problems related to developers' interruptibility. Gievska *et al.* [16] devised an interruptibility model to mediate human interruptions by a computer. In a study involving 24 knowledge workers (but not software developers), they showed, through the application of their model, that reducing interruptions can increase the perceived quality of work while decreasing frustration. For the specific case of software developers, Iqbal and Bailey [19] proposed a system that would postpone possible causes of disturbance to a more apt time based on cognitive theory. In a study with six professionals, they showed that their approach reduces frustrations while yielding to faster reaction time. More recently, Züger *et al.* [42] developed a physical device that would signal to the surrounding environment (*e.g.,* co-workers) the best moment to disturb (or not) a developer based on her computer interaction data. They carried out a field study with 449 participants (*i.e.,* knowledge workers, 119 of which working in software engineering-related activities), showing increased awareness about the disrupting effects of such kind of disturbance.

The studies above show that software developers' interruptibility is a topic worth investigating. However, there is only one study that focuses to some extent to noise as a source of interference with software developers' work. DeMarco and Lister related noise and other environmental factors (*e.g.,* space) to software developers' performance [12]. In particular, in a study with 166 professionals working on a benchmarking task, they showed that a quiet and commodious workplace could improve productivity (*e.g.,* time to complete the task) by a factor of 2.6. There are a number of differences between the study by DeMarco and Lister and that we present in this paper. The main differences can be summarized as follows: we conducted two studies in controlled conditions (*e.g.,* noise was measured in our case, while in the study by DeMarco and Lister participants provided their perception on the noise level in their workplace) on two kinds of software engineering tasks (SE task/s from here onwards) and we quantitatively assessed the effect of noise on performances (*i.e.,* achieved comprehension of functional requirements and capability to fix faults in source code). An additional difference is related to the programming language of the used experimental object (COBOL vs Java).

### 2.2 Background

We highlight the reference theories defined to explain and predict noise effects on individuals' performance.

*2.2.1 Arousal Theory.* To explain noise effects, Broadbent [6] invoked an arousal induced attentional narrowing mechanism in the individuals. In Broadbent's theory, noise increases arousal of an individual, which decreases his/her breadth of attention. At relatively lower levels of arousal, individuals exclude task-irrelevant cues, and thus the attentional narrowing facilitates performance. However, beyond a certain arousal "optimal" level, individuals' performance is impaired because increases in arousal might cause increased narrowing so that task-relevant cues are excluded. Arousal theory [6] predicts that more demanding tasks should have lower levels of optimum arousal, and thus these tasks should yield the greatest performance decrements in the presence of noise. Hence, cognitive tasks should suffer greater magnitudes of performance impairment with respect to less demanding tasks (*e.g.,* psychomotor ones). The intensity of noise and duration of noise exposure influence the arousal levels; *i.e.,* higher intensities and longer durations should cause greater negative effects on performance. Concerning noise schedule, intermittent noise should impair performance more than continuous one. In summary, in the Broadbent's theory noise effects should vary according to the kind of task and the noise intensity, duration, and schedule.

*2.2.2 Composite Theory.* Poulton's [29] theory predicts that noise effects should degrade individuals' performance only for those conditions in which inner speech[1] is masked. The gains in individuals' performance in continuous noise early in the task occur because the increase in arousal compensates for the detrimental effects of masking. However, with time on task, arousal decreases and thus masking effects dominate. The way in which arousal affects performances is different between composite and arousal theories. Noise intensity could also affect performance. In summary, noise effects should be similar across task and noise kind, but moderating effects are expected for intensity, duration, and schedule.

*2.2.3 Maximal Adaptability Theory.* In the maximal adaptability model [18], stress (noise is a source of stress) can be accounted for in three loci. *Input* represents all objective environmental and task factors that affect performance (*e.g.,* noise), *adaptation* concerns the capacity of the individual to cope with demands intrinsic to an environment (*e.g.,* physiological coping responses), and *output* refers to the individual's response about the task environment.

[1] Also referred to as verbal thinking, inner speaking, covert self-talk, internal monologue, and internal dialog. Inner speech is thinking in words and also refers to the semi-constant internal monologue some individuals have with themselves at either conscious or semi-conscious level [28].

**Table 1: Summary of the experiments.**

| Characteristic | Exp1 | Exp2 |
|---|---|---|
| Schedule | 11:30 on 2016/12/12 | 11:30 on 2017/1/31 |
| Kind of SE task | Comprehension of functional requirements | Fault fixing in source code |
| SE task duration | (2x) 30 minutes | (2x) 60 minutes |
| Experimental objects | M-Shop and Theater | LaTazza and AveCalc |
| Participants | 55 Undergraduate students in Computer Science | 42 Undergraduate students in Computer Science |
| Group1/Group2 size | 28/27 | 21/21 |
| Experiment design | AB/BA crossover design | AB/BA crossover design |
| Investigated RQ | RQ1 | RQ2 |

The output of a task depends on the characteristics of individuals, and it might be directly affected by noise. As for adaptation, noise can impair the capacity through the masking or distortion of task-relevant auditory information. According to the maximal adaptability theory, individuals can adapt to a quite broad range of stress magnitudes. However, there is a threshold of dynamic instability in which adaptation fails, and thus performance decreases. The maximal adaptability theory predicts that performance on more resource-demanding cognitive tasks, in case of noise, should be more impaired than performance on motor or perceptual tasks. For higher noise intensities and longer noise durations, there should be a greater performance impairment. Concerning the kind of noise, speech noise should be more disruptive than non-speech, especially in cognitive tasks. Also, noise schedule could affect performance. In summary, the maximal adaptability theory predicts that noise effects should vary as a function of task and noise kind, and schedule, duration, and intensity.

*2.2.4 Empirical Evidences from Noise Effects on Performance.* Arousal, composite, and maximal adaptability theories predict similar results on noise effects for certain variables (*e.g.,* noise intensity, duration, and schedule), but different results for others (*e.g.,* kind of task and noise). Szalma and Hancock [37] have recently conducted a meta-analysis on noise effects on individuals' performances. Results confirm only in part the predictions of the three reference theories, and in some cases are inconsistent with such predictions. That is, the meta-analytic results confirm that noise effects varied as a function of the kind of noise and task, and noise intensity, duration, and schedule. However, Szalma and Hancock reported that shorter durations have greater detrimental effects on performance than longer durations.

## 3 STUDY DESIGN

In Table 1, we summarize the main characteristics of our study, which comprises two controlled experiments. We refer to these experiments as Exp1 and Exp2 (see the second and third columns of Table 1), respectively. Exp1 was conducted on 2016/12/12, while Exp2 on 2017/1/31. In both experiments, we investigated whether noise affects performance when carrying out some SE tasks. The participants in Exp1 had to comprehend functional requirements of two software systems (*i.e.,* M-Shop and Theater as shown in the fourth row of Table 1) exposing them or not to noise. Similarly, the participants in Exp2 had to fix faults in two Java programs (*i.e.,* LaTazza and AveCalc as shown in Table 1).

To perform our study, we followed the guidelines by Juristo and Moreno [21], and Wohlin *et al.* [41]. We present the design of our study according to Jedlitschka *et al.*'s guidelines [20].

### 3.1 Goals

The goal of our study, according to the Goal Question Metrics (GQM) template [4], is:

**Analyze** noise **for the purpose of** evaluating its effect **with respect to** the performances in comprehending functional requirements and in fixing faults in Java source code **from the point of view of** the researcher **in the context of** final-year undergraduate students in Computer Science.

Therefore, we investigated the following research questions:

**RQ1** Does noise worsen software engineers' performance in comprehending functional requirements?

**RQ2** Does noise worsen software engineers' performance in fixing faults in source code?

### 3.2 Experimental Units

The participants in our experiments were final-year undergraduate students in Computer Science at the University of Basilicata. They had programming experience in Java and basic knowledge of software design, development, and testing. The participants attended the Software Engineering (SE) course in which we conducted both experiments as optional laboratory exercises. This course taught advanced elements concerning: software development processes, requirements specification, software design, maintenance, and testing. During the SE course, the participants carried out homework and classwork on requirements specification and bug fixing to increase their technical maturity on the topics covered in the course.

To encourage the participation in our study, we rewarded the students who took part in Exp1 with a bonus, *i.e.,* one point on their final mark in the SE course. Not all the participants in Exp1 took part in Exp2; a subset of the participants in the first experiment took part in the second one too. Students who took part in both experiments received two points of bonus. We communicated to the participants that their performance in the experiments would not affect their grade, and that the collected data would be shared anonymously and used for research purposes only. The participation in both experiments was on voluntary basis (*i.e.,* in no case we obliged students to participate in the experiments).

### 3.3 Experimental Material

As for Exp1, the chosen experimental objects where:

- **M-Shop**—A system for managing the sales in a music shop.
- **Theater**—A system for managing the reservation of tickets in a theater.

For each of these systems, one functional requirement with the corresponding models (*i.e.,* functional model, analysis object model, and dynamic model) was selected from its requirements analysis

| | |
|---|---|
| Q5. | Based on the furnished models, the primary actor can: (Mark the right answer/s)<br>□ Select any album<br>□ Modify an album<br>□ Get the available copies for an album<br>□ See the details of an album |

**Figure 1: A sample comprehension question for M-Shop.**

| Start Time (hh:mm): | End Time (hh:mm): |
|---|---|
| ID | 2 |
| Title | Wrong product selection |
| Description | If you select "supply of small bags" and then "Coffee" to buy a supply of coffee, arabica coffee is wrongly bought instead of coffee. The problem does not occur when buying a supply of arabica coffee. |

**Figure 2: A sample bug report for LaTazza.**

specification. In particular, the selected functional requirement for M-Shop was "Search Album by Singer," while for Theater was "Buy Theater Ticket." We chose these systems, and we selected these functional requirements because they were previously used in a family of controlled experiments to assess whether the comprehension of functional requirements was influenced by the use of dynamic models (represented through UML sequence diagrams) [1]. The authors, who conducted this family of experiments, administered the participants in the control group with the functional model and analysis object model associated with the selected functional requirement. The participants in the treatment group were administered with the same models as the control group plus the dynamic models (*i.e.,* UML sequence diagrams). To evaluate the comprehension of functional requirements, the authors asked the participants to fill out comprehension questionnaires. Assessing comprehension of software artifacts (*e.g.,* models or source code) through questionnaires is common in SE experiments (*e.g.,* [22, 32]). We exploited in Exp1 the experimental material, *i.e.,* models and questionnaires, the authors [1] made available on the web and they administered to the participants in the treatment group. This design choice should not affect the results because the participants who accomplished the task in noise conditions were provided with the same material as the participants who accomplished the task in normal conditions.

As for Exp2, we chose the following experimental objects:

- **LaTazza**—A Java desktop application for managing the sale and the supply of small-bags of beverages (*e.g.,* tea) for a coffee maker. Its source code had 18 classes and 1,215 LOC (*i.e.,* Lines of Code).
- **AveCalc**—A Java desktop application for managing the exams of a student during its university career. Its source code had 33 classes and 1,388 LOC.

LaTazza and AveCalc were used in other empirical studies (*e.g.,* [31, 34]). In particular, we exploited the source code of the experimental objects used in the three experiments by Ricca *et al.* [31] (the source code they administered to the control group was the same as the treatment group). To assess the performance in fixing faults in source code, we provided the participants with bug reports and asked them to fix the faults that such bug reports described. We exploited the bug reports Scanniello *et al.* [34] defined on the experimental objects by Ricca *et al.* [31] and then used in their family of controlled experiments (the bug reports administered to participants were the same in both treatment and control groups). Similar to Exp1, we used the experimental material (*i.e.,* source code and bug reports) the authors [31, 34] made available on the web. It is worth mentioning that assessing the performance in fixing faults in source code with a different instrumentation tool (*e.g.,* picking the correct fix for a bug from a multiple-choice question) would have decreased the realism of the fault fixing tasks. Conversely, the used instrumentation tool allowed reducing threats to external validity.

We used experimental materials defined by different researchers to mitigate experimenters' expectancies biases. We made both experimental material and raw data available on the web.[2]

## 3.4 Tasks

The participants in Exp1 had to perform the following tasks:

(1) **Comprehension task 1**—We provided each participant with the models associated with the functional requirement "Search Album by Singer" of M-Shop. To evaluate the comprehension of such a requirement, we asked the participants to fill out a comprehension questionnaire consisting of 11 closed-ended questions. Each question admitted one or more right answers. In Figure 1, we report a sample question of the comprehension questionnaire of M-Shop, which admitted two right answers: "Get the available copies for an album" and "See the details of an album."
(2) **Comprehension task 2**—We asked the participants to perform the same task as the previous one, but the experimental object was Theater. In particular, we gave each participant the models associated with the functional requirement "Buy Theater Ticket." Then, we asked to fill out a comprehension questionnaire similar (*e.g.,* it comprised 11 closed-ended questions) to that used in the previous task.

As for Exp2, the participants had to perform the following tasks:

(1) **Bug fixing task 1**—We provided the participants with the source code (no test cases were given) and mission (*i.e.,* problem statement) of LaTazza. We also gave them six bug reports, one for each fault the participants had to fix in the source code of LaTazza. A bug report contained the ID of the corresponding bug, as well as a title and a description. In Figure 2, we show a sample bug report for LaTazza. When a participant fixed a fault, he/she had to indicate the portion of source code he/she modified to fix such a fault, *i.e.,* his/her patch. In particular, the participant had to insert two Java comments to delimit the modified code: `/*patch <fault_ID> start*/` just before the patch, and `/*patch end*/` at the end of the patch. This procedure is the same as used by Scanniello *et al.* [34], where further details on the bug reports and the faults are available and that we have not reported here for space reason and scant relevance. The participants had to also write down when they start tackling each bug and when it was fixed (see Figure 2).
(2) **Bug fixing task 2**—We asked the participants to perform the same task as the previous one but on AveCalc. Therefore, we provided them with the source code and mission of AveCalc together with six bug reports. To indicate the patches, the participants had to use Java comments as done in the previous task.

[2] www2.unibas.it/sromano/downloads/NoiseExpsReplicationPackage.zip

### 3.5 Hypotheses, Parameters, and Variables

The participants who had to perform the tasks (*i.e.,* comprehension of functional requirements or bug fixing in source code) in the presence of noise comprised the treatment group; while those who worked on the tasks in normal conditions (*i.e.,* they were not exposed to noise) comprised the control group. Therefore, in each experiment *Condition* is the main independent variable (also known as main factor or manipulated factor or main explanatory variable). Condition is a nominal variable that assumes two values: NOISE (*i.e.,* participants working in noise conditions) and NORMAL (*i.e.,* participants working in normal conditions).

To quantify software engineers' performance in comprehending functional requirements, we evaluated participants' answers to each comprehension questionnaire by using two strategies. The first one was an information retrieval-based strategy [24]. It consists in computing precision ($P_c$) and recall ($R_c$) of the answers given by the participant $s$ as follows:

$$P_c = \frac{\sum_{i=1}^{n} |answers_{s,i} \cap oracle_i|}{|\sum_{i=1}^{n} answers_{s,i}|} \quad R_c = \frac{\sum_{i=1}^{n} |answers_{s,i} \cap oracle_i|}{|\sum_{i=1}^{n} oracle_i|}$$

where $answers_{s,i}$ is the set of answers the participant $s$ provided for the question $i$, while $oracle_i$ is the set of correct expected answers (*i.e.,* the oracle) for the question $i$. We indicate with $n$ the number of questions (*i.e.,* 11 for both M-Shop and Theater). From a practical perspective, $P_c$ and $R_c$ estimate correctness and completenesses of the given answers. To get a single measure that represents a trade-off between correctness and completenesses, we compute the balanced F-measure [24] between $P_c$ and $R_c$ as follows:

$$F_c = \frac{2 * P_c * R_c}{P_c + R_c}$$

This metric assumes values in the interval [0, 1], where 1 means that the participant answered very good the questions of the comprehension questionnaire, so assuming that his/her comprehension of functional requirements was very good. Conversely, a value close to 0 means that he/she answered very bad (*i.e.,* functional requirements were scarcely comprehended). This information retrieval-based strategy is the same as used by Abrahão *et al.* [1]. The second strategy we adopted to quantify performance in comprehending functional requirements was inspired to that used by Kamsties *et al.* [22]. In particular, for each participant $s$, we computed the variable *count* as follows:

$$count_{s,i} = \begin{cases} 1 & if\ answers_{s,i} = oracle_i \\ 0 & otherwise \end{cases}$$

*count* assumes 1 as the value if and only if the set of answers provided by the participant $s$ to the question $i$ corresponds to the oracle for the question. We quantified performance in comprehending functional requirements by means of the following metric:

$$Avg = \frac{\sum_{i=1}^{n} count_{s,i}}{n}$$

where $n$ is the number of questions. $Avg$ assumes values in the interval [0,1], where 1 is the best possible value. In other words, the higher the $Avg$ value, the better the comprehension of functional requirements is. Unlike $F_c$, $Avg$ does not take into account partial answers. The use of two metrics (*i.e.,* $F_c$ and $Avg$) to assess the comprehension of functional requirements allowed us to mitigate possible construct validity threats (*i.e.,* mono-method bias) [41].

To measure software engineers' performance in fixing faults in source code, we relied on the information retrieval-based strategy used Scanniello *et al.* [34]. We estimated correctness and completeness of the faults the participant $s$ fixed by means of the precision ($P_f$) and recall ($R_f$) measures, respectively:

$$P_f = \frac{\#\ faults\ correctly\ fixed_s}{\#\ faults\ fixed_s} \quad R_f = \frac{\#\ faults\ correctly\ fixed_s}{\#\ faults\ present}$$

where $\#\ faults\ fixed_s$ is the count of bug the participant $s$ fixed both correctly and incorrectly. We knew this information because each participant had to delimit his/her patch when fixing a given fault (see Section 3.4). To get $\#\ faults\ correctly\ fixed_s$, we first inspected the patches provided by the participant $s$, then we run an acceptance test suite for each bug. If this suite passed, the corresponding bug was correctly fixed. $\#\ faults\ present$ is the number of bug the participants had to fix (*i.e.,* six for both LaTazza and AveCalc).

To get a trade-off between correctness and completeness of fixed faults, and thus estimate performance in fixing fault in source code, we computed the balanced F-measure as follows:

$$F_f = \frac{2 * P_f * R_f}{P_f + R_f}$$

A value for $F_f$ close to 0 means that the performance in fixing faults was very bad (*i.e.,* existing bugs were, for the most part, either not fixed or incorrectly fixed), while a value close to 1 means that the performance in fixing faults was very good.

According to our research questions, we formulated the following null-hypotheses:

**Hn1** Noise *does not significantly affect* software engineers' performance in comprehending functional requirements.

**Hn2** Noise *does not significantly affect* software engineers' performance in fixing faults in source code.

Given the literature regarding the effect of noise (*e.g.,* [6, 18, 29, 37]), we expect that noise might have detrimental effect on software engineers' performance. Therefore, previous knowledge allows us to formulate the one-tailed alternative hypotheses: *noise significantly and negatively affects software engineers' performance in [comprehending functional requirements | fixing faults in source code].*

### 3.6 Experiment Design

In Figure 3, we graphically describe the overall design of our empirical investigation. Before Exp1 took place, the participants had to fill out a pre-questionnaire to gather their demographic information. We used Google Forms to create this questionnaire and left the participants the possibility to fill it out from 2016/11/22 to 2016/11/29. The gathered information on the participants allowed us to characterize the context of our study better.

For both experiments, the used design was an AB/BA crossover. AB/BA crossover designs have two treatments (*i.e.,* A and B) and two periods (*i.e.,* the times at which each treatment is applied). Participants are split into two experimental groups and administered with every treatment only once. In AB/BA crossover designs, the groups are the sequences, *i.e.,* the order in which treatments are administered to participants [38].

Figure 3 also shows how the AB/BA crossover design was applied to both Exp1 and Exp2. In our case, the treatments A and B were

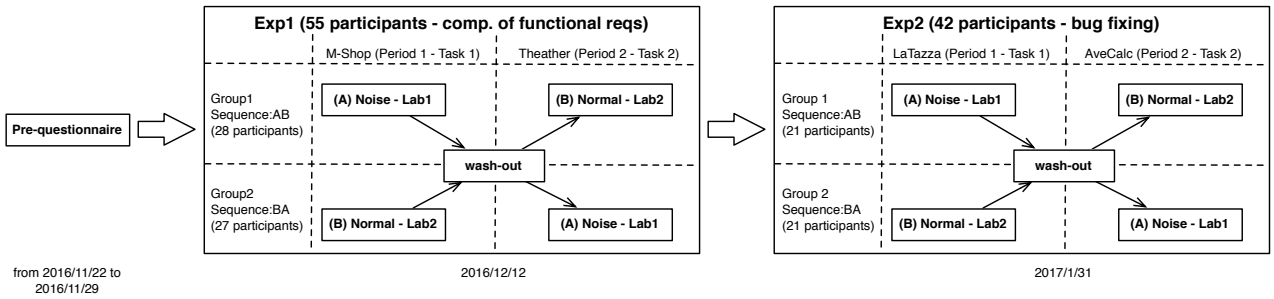



**Figure 3: Summary of the design.**

NOISE and NORMAL, respectively. The participants were randomly assigned to the experimental groups:

- **Group1**—It comprises participants assigned to the sequence AB (*i.e.,* NOISE is applied in the first period, while NORMAL in the second).
- **Group2**—It comprises participants assigned to the sequence BA (*i.e.,* NORMAL is applied in the first period, while NOISE in the second).

As for Exp1, the number of participants in Group1 and Group2 was 28 and 27, respectively; while 21 and 21 in Exp2. Independently from the treatment, the participants in Exp1 performed comprehension task 1 (on M-Shop) in the first period, while comprehension task 2 (on Theater) in the second. The participants in Exp2 carried out bug fixing task 1 (on LaTazza) in the first period, while bug fixing task 2 (on AveCalc) in the second. Group1 and Group2 carried out each task (*e.g.,* comprehension task 1) at the same time.

Comprehension tasks lasted 30 minutes each, while bug fixing tasks lasted one hour each. We fixed the time to accomplish the tasks on the basis of the experimental data reported by Abrahão *et al.* [1] and Scanniello *et al.* [34]. The use of tasks with different durations allowed studying the effect of duration on performance. The strategy we used to define performance (in both the experiments) was *time fixed*, namely the number of successful steps within the time limit defines performance [5].

NOISE was always applied in the Lab1 research laboratory, while NORMAL always in Lab2 (see Figure 3). We took this design decision to mitigate possible conclusion validity threats related to the experimental settings.

We added a 30-minute wash-out period between Period 1 and Period 2 (see Figure 3). The rationale behind the use of wash-out periods (as suggested in medicine studies) is to leave sufficient time for the effect of a treatment to recede completely, and thus possibly neutralize carryover effects.[3] However, it can be difficult to determine how long a wash-out period should be to recede completely the effect of a treatment [11]. That is, the required length of wash-out period for a given treatment is usually unknown before the study takes place [15]. Therefore, carryover effects could also occur in the presence of wash-out periods if they are not long enough. Therefore, we analyzed whether carryover effects were present in our experiments (see Section 3.8).

## 3.7 Experiment Setting

The Directive 2003/10/EC[4] of the European Parliament lays down minimum health and safety requirements regarding the exposure of workers to the risks arising from noise. In particular, if the noise exposure level[5] ($L_{EX}$) is greater than or equal to 85 dB, workers shall wear individual hearing protectors; thus values that match or exceed such a limit are considered harmful to health. The participants administered with the NOISE treatment had to accomplish the tasks with a value of $L_{EX}$ (*i.e.,* 82 dB) close but inferior to the limit mentioned above. Thus individual hearing protectors were not required. The kind of noise was speech because it is the major type of practical distractive noise in workplaces with open-office plans [25, 37]. As for the participants working in normal conditions, they performed the experimental tasks in a laboratory (*i.e.,* Lab2) far from road traffic and other sources of noise (the measured $L_{EX}$ value was equal to 42 dB, namely a common value of noise level for quiet office workplaces [25]). We chose the exposure value for the NOISE treatment on the basis of the results from the study by Szalma and Hancock: low-intensity noise is more debilitating than high-intensity noise for those studies in which quiet is the control condition [37]. Therefore, if we found a significant effect of noise, we would expect a higher effect of noise for lower $L_{EX}$ values.

The NOISE treatment was administered in a laboratory (*i.e.,* Lab1) equipped with a sound system. A (ceiling) speaker was present on each workstation used by each participant. The distance between each speaker and each workstation was always the same. Before the experiments took place, for any workstation we measured the noise level chosen for the experimental setting with a sound meter. In particular, we placed the sound meter 45 cm above the desk. Such a measurement allowed us to grasp, with some approximation,what a participant would have heard during the tasks. We measured the noise levels also during the experiments. Lab1 was far from road traffic (*i.e.,* the most dominant source of environmental noise [3]) and other sources of noise that might have interfered with the noise reproduced by the sound system.

Furniture (*e.g.,* the kind of desk or chair) in Lab1 was the same as in Lab2. These laboratories were equipped with the same PCs (*i.e.,* same hardware and software configurations) and the environmental configurations were similar (*e.g.,* similar light level) as well as the design (*i.e.,* open-office). This is because we wanted to mitigate environmental factors that might affect results in an unexpected way. Similar to Lab1, Lab2 was far from sources of noise.

[3] Carryover is an internal validity threat of crossover designs. It occurs when a treatment is administered before the effect of another previously administered treatment has completely receded [38].

[4] eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02003L0010-20081211

[5] It is the time-weighted average of the noise exposure levels.

**Table 2: Results from the pre-test to check the presence of carryover effects (in bold p-values less than $\alpha$).**

| Experiment | Variable | p-value |
|---|---|---|
| Exp1 | $F_c$ | 0.5179 |
| | $Avg$ | 0.4312 |
| Exp1* | $F_c$ | 0.2432 |
| | $Avg$ | 0.1809 |
| Exp2 | $F_f$ | **0.0358** |

## 3.8 Analysis Procedure

We used the following analysis procedure for each experiment and each dependent variable. According to Wellek and Blettner's study [40], we perform a pre-test to check the presence of a carryover effect. Let $X_{1i}$ and $X_{2i}$ be the dependent variable values (*e.g.,* $F_c$) for the participant $i$ of Group1 (*i.e.,* sequence AB) in the first and second periods, respectively. Similarly, let $Y_{1j}$ and $Y_{2j}$ be the dependent variable values for the participant $j$ of Group2 (*i.e.,* sequence BA) in the first and second periods, respectively. For each participant $i$ of Group1 and each participant $j$ of Group2, we compute the within-participant sums of the dependent variable values in both periods as follows: $C_i(X) = X_{1i} + X_{2i}$ and $C_j(Y) = Y_{1j} + Y_{2j}$. If data are normally distributed, we run an unpaired two-sided t-test to verify the null-hypothesis (*i.e.,* the expected mean values of within-participant sums are the same [40]). In case data are not normally distributed, we run a two-sided Mann-Whitney U test [8] (also known as Wilcoxon rank-sum test). The Mann-Whitney U test is a non-parametric alternative to the t-test [41]. If the t-test/Mann-Whitney U test does not reject the null-hypothesis, the carryover effect is not statistically significant.

If carryover effect is not statistically significant, we performed the following steps:

(1) We computed descriptive statistics and build boxplots for each dependent variable.
(2) We tested whether the effect of noise was statistically significant on the dependent variable. To this end, for each participant $i$ of Group1 and each participant $j$ of Group2, we computed the within-participant differences of the dependent variable values in both periods [40]: $D_i(X) = X_{1i} - X_{2i}$ and $D_j(Y) = Y_{1j} - Y_{2j}$. Then, if the data were normally distributed, we run an unpaired two-sided t-test, where the tested null-hypothesis is: are the expected mean values of within-participant differences the same [40]? We run a two-sided Mann-Whitney U test, if data are not normally distributed.

If carryover effect is statistically significant for a given depended variable, we discarded the second period [38]. That is, we analyzed the first period as follows:

(1) We computed descriptive statistics and build boxplots.
(2) We tested if the effect of noise was statistically significant on the dependent variable values by means of an unpaired two-sided t-test, if the data were normally distributed, or a two-sided Mann-Whitney U test otherwise.

To check normality of data, we used the Shapiro-Wilk test [35] (Shapiro test, from here onwards).

As it is customary with tests of statistical significance, we accept a probability of 5% of committing Type-I error (*i.e.,* $\alpha = 0.05$).

**Table 3: Some descriptive statistics grouped by experiment, variable, and condition.**

| Experiment | Variable | Statistic | NORMAL | NOISE |
|---|---|---|---|---|
| Exp1 | $F_c$ | Median | 0.6923 | 0.7143 |
| | | Mean | 0.6781 | 0.6833 |
| | | Std | 0.1268 | 0.1398 |
| | $Avg$ | Median | 0.5455 | 0.5455 |
| | | Mean | 0.5025 | 0.5256 |
| | | Std | 0.1296 | 0.1552 |
| Exp1* | $F_c$ | Median | 0.6923 | 0.7143 |
| | | Mean | 0.687 | 0.6949 |
| | | Std | 0.118 | 0.1382 |
| | $Avg$ | Median | 0.5455 | 0.5455 |
| | | Mean | 0.5065 | 0.5281 |
| | | Std | 0.124 | 0.1646 |
| Exp2 | $F_f$ | Median | 0.5 | 0.2857 |
| | | Mean | 0.4814 | 0.3149 |
| | | Std | 0.2969 | 0.1983 |

Since 42 out of 55 participants took part in both the experiments, we also studied the performances of the participants in Exp1 who also participated in Exp2. We refer to this subset as Exp1*, where Group1 and Group2 comprised 21 participants each. The analysis procedure for Exp1* was the same as that used for Exp1 and Exp2. The goal of this further analysis was to see if the results from Exp1 are confirmed or not by those from Exp1*. In case of confirmation, we can state that the results from Exp1 were not due to any characteristic of the participants who did not take part in Exp2.

# 4 RESULTS

In this section, we report the results of our analysis.

## 4.1 Carryover Analysis

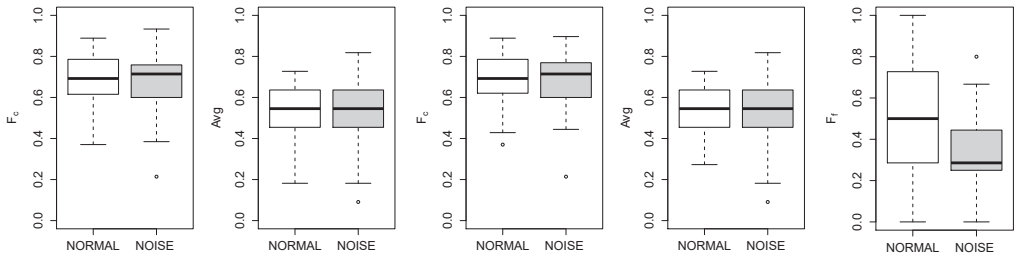
The Shapiro test suggested that the data were normally distributed in each experiment and for each dependent variable (*i.e.,* p-values were greater than $\alpha$), thus we applied a t-test to check the presence of a carryover effect. The obtained p-values are reported in Table 2. The results indicate that the carryover effect is not statistically significant for any dependent variable in Exp1 and Exp1*. The carryover effect is statistically significant for $F_f$ in Exp2 (p-value=0.0358); thus we analyzed only the first period.

## 4.2 Descriptive Statistics and Boxplots

In Table 3, we summarize descriptive statistics of the dependent variable values the participants obtained in each experiment grouped by treatment (*i.e.,* NORMAL or NOISE). These values are also graphically summarized using the boxplots in Figure 4.

As for Exp1, the descriptive statistics suggest that there is not a huge difference between the participants administered with NORMAL and NOISE with respect to $F_c$ (0.6781 vs 0.6833 on average) and $Avg$ (0.5025 vs 0.5256 on average). The boxplots in Figure 4.a confirms this similarity. This result seems to be confirmed in Exp1*. On average, the $F_c$ values are 0.687 for NORMAL and 0.6949 for NOISE, whereas the $Avg$ values are 0.5065 and 0.5281, respectively.

As for Exp2, the descriptive statistics of the $F_f$ values seem to indicate that there is a difference between the NORMAL and NOISE treatments (0.4814 vs 0.3149 on average). The boxplots, shown in Figure 4.c, seem to confirm this result.




(a) Exp1 (b) Exp1* (c) Exp2

**Figure 4: Boxplots for: $F_c$ and $Avg$ in Exp1 (a), $F_c$ and $Avg$ in Exp1* (b), and $F_f$ in Exp2 (c).**

**Table 4: Results from the testing of $Hn1$ and $Hn2$ (p-values less than $\alpha$ are reported in bold).**

| Hn | Experiment | Variable | p-value | Effect size |
|---|---|---|---|---|
| Hn1 | Exp1 | $F_c$ | 0.7309 | - |
| | | $Avg$ | 0.3119 | - |
| | Exp1* | $F_c$ | 0.7116 | - |
| | | $Avg$ | 0.3782 | - |
| Hn2 | Exp2 | $F_f$ | **0.024** | medium (0.4014) |

**Table 5: Some descriptive statistics for $F_f$ when tanking into account only the first 30 minutes.**

| Statistic | NORMAL | NOISE |
|---|---|---|
| Median | 0.2857 | 0.2857 |
| Mean | 0.3141 | 0.1741 |
| Std | 0.2163 | 0.2024 |

## 4.3 Hypotheses Testing

*4.3.1 Hn1—comprehension of functional requirements.* As for Exp1, we used the t-test for $F_c$ because the data were normally distributed, while we used the Mann-Whitney U test for $Avg$ because the Shapiro test indicated that the data were not normally distributed (*i.e.,* p-value = 0.0167 for Group2). As for Exp1*, we used the t-test for both $F_c$ and $Avg$ (the Shapiro test always returned p-values greater than $\alpha$). For both Exp1 and Exp1*, and any dependent variable, we could not reject $Hn1$ (see Table 4).

*4.3.2 Hn2—fault fixing in source code.* The Mann-Whitney U test—the Shapiro test returned a p-value equal to 0.0229 for NOISE—allowed us to reject $Hn2$ with respect to $F_f$ (p-value = 0.024, see Table 4). Thus, noise has a significant and negative effect on software engineers' performance in fixing faults in source code. To measure the magnitude of the difference, we used the Cliff's $\delta$ effect size [7]. This kind of effect size is used when data are not normally distributed or the normality assumption is discarded. As shown in Table 4, the effect of noise on fault fixing tasks was medium.[6]

## 4.4 Further Analysis

We analyzed the data from Exp2 by taking into account the first 30 minutes (first period). This was possible because we knew when each participant started tackling any bug and when it was fixed (see Section 3.4). We perform this further analysis to exclude noise duration (Exp1 and Exp2 lasted 30 and 60 minutes, respectively) from the possible causes behind the lack of statistical significant difference in Exp1 (and Exp1*), with respect to Exp2.

In Table 5, we report some descriptive statistics for $F_f$ with respect to the first 30 minutes of Exp2. These descriptive statistics indicate that, on average, who fixed faults in the presence of noise had worse performance than who worked in normal conditions (the mean values of $F_f$ are 0.1741 for NOISE and 0.3141 for NORMAL). To check that this difference was statistically significant, we ran a two sided Mann-Whitney U test (the data were not normally distributed for both NOISE and NORMAL as the Shapiro test suggested). The returned p-value (0.0281) indicate that the participants exposed to noise for 30 minutes had significantly worse performances in fixing fault in source code than the participants not exposed. The effect of noise was medium (the Cliff's $\delta$ effect size value was 0.3605). Thus, we can exclude that different noise durations are behind the lack of statistical significant difference in Exp1 (and Exp1*).

# 5 DISCUSSION

In the next subsections, we first discuss the results by linking them to RQ1 and RQ2. Then, we delineate practical implications from the obtained results and future directions for our research. We conclude discussing threats that could affect the validity of our results.

## 5.1 Linking Results to Research Questions and Overall Discussion

Results from Exp1 and Exp1* suggest that noise does not significantly affect the comprehension of functional requirements. Thus, we cannot positively answer RQ1. On the other hand, we observed that noise negatively affects faults fixing (also restricting the analysis to the first 30 minutes in Exp2, see Section 4.4). Thus, we can positively answer RQ2: "noise worsens software engineers' performance in fixing faults in source code." This outcome suggests that noise duration is not the cause of the lack of statistically significant difference in Exp1 (and Exp1*), but it is the kind of task. Fixing faults in source code seems to be more vulnerable to noise than comprehending functional requirements. According to the meta-analytic results by Szalma and Hancock's [37], we can then speculate that fixing faults is more resource-demanding than comprehending functional requirements. Concluding, it seems that there are tasks in software engineering that are more resource-demanding than others and noise seems to negatively affect the execution of these tasks.

[6] According to Romano *et al.* [33], the Cliff's $\delta$ effect size is: "negligible" if $|\delta| < 0.147$, "small" if $0.147 \leq |\delta| < 0.33$, "medium" if $0.33 \leq |\delta| < 0.474$, or "large" otherwise.

## 5.2 Implications and Future Directions

We focus here on practitioner and researcher perspectives.

- Comparing our results on the comprehension of functional requirements to those reported by Abrahão *et al.* [1] (where the same experimental material and dependent variable $F_c$ were used), we observe that our participants (*i.e.,* final-years undergraduate students) performed worse than Ph.D. students (on average, 0.6781 is the $F_c$ value in Exp1, while 0.727 is the one achieved by the Ph.D. students), but better than professionals (0.6781 vs 0.631, on average). Thus, we postulate that the participants in Exp1 had an adequate level of familiarity with the used modeling notation, so allowing us to assume that they are representative of professionals. Therefore, rather than replicating the first experiment with professionals, the researcher could be interested in investigating whether different noise intensities (*i.e.,* levels) affect software engineers' performance in comprehending functional requirements. The researcher could also be interested in varying the modeling methods (*i.e.,* UML) and assessing if the comprehension of functional requirements is still not vulnerable to noise.
- We observed a statistically significant difference in fault fixing tasks when the participants were exposed or not to noise. This finding is relevant for the practitioner because noisy workspaces (*e.g.,* those with open-office plans) could cause a reduction in the performance of software engineers that have to fix faults in Java code. In other words, a penny saved on the workspace could be paid with interest later. However, we believe that caution is needed concerning this finding and we advise future work. For example, the researcher could investigate if Exp2 results also hold for professionals. Other directions for future work could consist in varying the noise intensity and type.
- Fixing faults in source code seems to be more vulnerable to noise than comprehending functional requirements. This finding is consistent with arousal and maximal adaptability theories [6, 18] and the results from the Szalma and Hancock's meta-analysis [37]. The researcher could be interested in studying which tasks are more (or less) vulnerable to noise. That is, some kinds of tasks (*i.e.,* those less resource-demanding) could be weakly affected by noise. Thus, software companies could save money by providing workspaces with open-office plan to the software engineers involved in these kinds of tasks. On the other hand, more resource-demanding tasks should be more vulnerable to noise. Thus, quiet workspaces (*e.g.,* those with closed-office plans) are advisable. This is clearly relevant for the practitioner.
- Given our results, we speculate that a 30-minute wash-out period is not enough as far as 1-hour fault fixing tasks is concerned. Our results provide a reference point for the duration of wash-out periods, which we did not have when we designed our experiments. This finding is of interest for the practitioner too. For example, our results suggest software engineers to take a break greater than 30 minutes when performing a fault fixing task in the presence of noise. Such a long wash-out period might be expensive for a software company. Therefore, it could be more advisable to provide software engineers with quieter workspaces.
- The presence of a significant carryover effect in Exp2 indicates a detrimental effects of noise on the performance, when fixing faults in Java code, even when participants are no longer exposed to noise. This is relevant for the practitioner. On the other hand, the researcher could be interested in further investigating on this matter using special conceived investigations.

## 5.3 Threats to Validity

We discuss threats that could affect our results by following the recommendations by Wohlin *et al.* [41].

*5.3.1 Internal Validity.* Social threats to validity might exist. The participants exposed to noise might be more motivated to accomplish the task (*e.g.,* due to arousal effect) than the participants working in normal condition [6, 18]. On the other hand, the participants exposed to noise might give up performing as they would do under normal conditions (*i.e., resentful demoralization*). This kind of threat is present when the data analysis is conducted on the first period (*i.e.,* when dealing with a statistically significant carryover).

The *selection* threat of letting volunteers take part in the experiments could influence the results since they could be more motivated than actual developers.

To prevent that participants exchanged information on the tasks (*i.e., diffusion or imitation of treatments*), we monitored them during the execution of each task and we took back all the material we gave them to accomplish the tasks. In addition, Group1 and Group2 performed the same task (*e.g.,* bug fixing on LaTazza) at the same time (see Section 3.6) in each experiment.

The use of an AB/BA design might affect internal validity. We dealt with this kind of threat by means of a wash-out period in each experiment. We studied if wash-out periods were long enough to neutralize carryover effect (Section 3.8). In case of carryover was present, we used the strategy suggested by Vegas *et al.* [38] (*i.e.,* taking into account only the first period in the data analysis).

*5.3.2 Construct Validity.* To deal with this kind of threat, we exploited metrics well known and adopted in the literature (*e.g.,* [1, 30, 34]). As far as Exp2 is concerned, we considered only one metric to assess participants' performances (*i.e., mono-method bias*).

Although we did not inform the participants about our research goals, they were aware of being part of an investigation on noise effect. Thus, there could be the risk of *hypotheses guessing*.

We informed students that achieved performance would not affect the SE course grades and gather data would be shared anonymously and in aggregated fashion (*i.e., evaluation apprehension*).

*5.3.3 Conclusion Validity.* The implementation of a treatment (*e.g.,* NOISE) might be not similar between different participants (*i.e., reliability of treatment implementation*). As shown in Section 3.7, we mitigate this kind of threat by implementing treatment/control as standard as possible over different participants.

*5.3.4 External Validity.* The participants were sampled by convenience. Therefore, generalizing the results to a different population (*e.g.,* professional software developers rather than students at the University of Basilicata) poses a threat of *interaction of selection and treatment*. Working with students also implies various advantages, such as: their homogeneous prior knowledge, the availability of a large number of participants [39] (55 and 42 participants in studies

like ours is appreciable), and the possibility to test experimental design and initial hypotheses [36].

The use of UML as modeling notation in Exp1 might threaten the generalizability of the results (*i.e., interaction of setting and treatment*). We opted for UML because it is a de-facto standard in software modeling and the students were familiar with it. Another threat of interaction of setting and treatment might exist due to the non-real-world experimental tasks used. The experimental tasks should equally affect the results of the participants when exposed or not to noise.

## 6 CONCLUSION

We present the results of an empirical evaluation constituted of two controlled experiments. Both experiments assess whether noise negatively influences software engineers' performances. In the first experiment, we asked 55 final-year undergraduate students in Computer Science to comprehend functional requirements. While performing this task, the participants in this experiment were exposed or not to noise. We asked these participants to take part in the second experiment (42 out of 55 agreed to participate in) where the task to be performed was fixing faults in Java source code. The results suggest that the participants had significantly worse performance in fixing faults in source code when exposed to noise, while no difference was observed in comprehending functional requirements. We conjecture that bug fixing is a more resource-demanding than comprehending functional requirements as it is more vulnerable to noise. Therefore, it seems that there are more resource-demanding tasks than others, and noise seems to negatively impact tasks that are more resource-demanding.

## REFERENCES


[1] S. Abrahão, C. Gravino, E. Insfran, G. Scanniello, and G. Tortora. 2013. Assessing the Effectiveness of Sequence Diagrams in the Comprehension of Functional Requirements: Results from a Family of Five Experiments. *IEEE Trans. on Softw. Eng.* 39, 3 (2013), 327–342.

[2] S. T. Acuna, N. Juristo, A. M. Moreno, and A. Mon. 2010. *A Software Process Model Handbook for Incorporating People's Capabilities* (1st ed.). Springer Publishing Company, Incorporated.

[3] European Environment Agency and Europæiske Miljøagentur. 2014. *Noise in Europe 2014.* European Environment Agency.

[4] V. Basili, G. Caldiera, and D. H. Rombach. 1994. *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering.* John Wiley and Sons. 528–532 pages.

[5] G. R. Bergersen, D. I. K. Sjøberg, and T. Dybå. 2014. Construction and Validation of an Instrument for Measuring Programming Skill. *IEEE Trans. on Softw. Eng.* 40, 12 (2014), 1163–1184.

[6] D.E. Broadbent. 1978. The current state of noise research: Reply to Poulton. *Psychological Bulletin* 85 (1978), 1052–1067.

[7] N. Cliff. 1996. *Ordinal methods for behavioral data analysis.* Psychology Press, New-York, USA.

[8] W. J. Conover. 1998. *Practical Nonparametric Statistics* (3rd ed.). Wiley.

[9] L.L. Constantine. 1995. *Constantine on Peopleware.* Yourdon Press.

[10] M. Csikszentmihalyi. 2009. *Creativity: Flow and the Psychology of Discovery and Invention (Harper Perennial Modern Classics).* HarperCollins e-books.

[11] A. Dean, M. Morris, J. Stufken, and D. Bingham. 2015. *Handbook of Design and Analysis of Experiments.* Chapman and Hall/CRC.

[12] T. DeMarco and T. Lister. 1985. Programmer Performance and the Effects of the Workplace. In *Proc. of the International Conference on Software Engineering.* IEEE, 268–272.

[13] T. DeMarco and T. Lister. 2013. *Peopleware: Productive Projects and Teams* (3rd ed.). Addison-Wesley Professional.

[14] F. Fagerholm and J. Münch. 2012. Developer experience: Concept and definition. In *International Conference on Software and System Process.* IEEE, 73–77.

[15] J. I. Gallin and F. P. Ognibene. 2007. *Principles and Practice of Clinical Research* (2nd ed.). Academic Press.

[16] S. Gievska, R. Lindeman, and J. Sibert. 2005. Examining the qualitative gains of mediating human interruptions during hci. In *Proc. of the International Conference on Human-Computer Interaction.*

[17] D. Graziotin, X. Wang, and P. Abrahamsson. 2014. Software Developers, Moods, Emotions, and Performance. *IEEE Software* 31, 4 (2014), 24–27.

[18] P. A. Hancock and J. S. Warm. 1989. A Dynamic Model of Stress and Sustained Attention. *Human Factors* 31, 5 (1989), 519–537.

[19] Shamsi T. Iqbal and Brian P. Bailey. 2007. Understanding and Developing Models for Detecting and Differentiating Breakpoints During Interactive Tasks. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 697–706.

[20] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. 2008. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering.* Springer London, 201–228.

[21] N. Juristo and A.M. Moreno. 2001. *Basics of Software Engineering Experimentation.* Kluwer Academic Publishers.

[22] E. Kamsties, A. von Knethen, and R. Reussner. 2003. A controlled experiment to evaluate how styles affect the understandability of requirements specifications. *Inf. Softw. Technol.* 45, 14 (2003), 955–965.

[23] K. Kuusinen, H. Petrie, F. Fagerholm, and T. Mikkonen. 2016. Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. In *Agile Processes in Software Engineering and Extreme Programming.* Springer International Publishing, 104–117.

[24] C. D. Manning, P. Raghavan, and H. Schtze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA.

[25] L. E. Maxwell. 2015. Noise in the office workplace. *Facility Planning and Management Notes* 1, 11 (2015).

[26] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. 2017. The Work Life of Developers: Activities, Switches and Perceived Productivity. *IEEE Trans. on Softw. Eng.* PP, 99 (2017), 1–1.

[27] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. 2014. Software Developers' Perceptions of Productivity. In *Proc. of the ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, 19–29.

[28] A. Morin, J. D. Runyan, and T. M. Brinthaupt. 2015. Editorial: Inner Experiences: Theory, Measurement, Frequency, Content, and Functions. *Frontiers in Psychology* 6 (2015), 1758.

[29] E.C. Poulton. 1979. Composite model for human performance in continuous noise. *Psychological Review* 86 (1979), 361–375.

[30] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, and M. Ceccato. 2010. How Developers' Experience and Ability Influence Web Application Comprehension Tasks Supported by UML Stereotypes: A Series of Four Experiments. *IEEE Trans. on Softw. Eng.* 36, 1 (2010), 96–118.

[31] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, M. Ceccato, and C. A. Visaggio. 2008. Are fit tables really talking?. In *Proc. of the ACM/IEEE International Conference on Software Engineering.* IEEE, 361–370.

[32] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano. 2014. Assessing the Effect of Screen Mockups on the Comprehension of Functional Requirements. *ACM Trans. Softw. Eng. Methodol.* 24, 1, Article 1 (2014), 1:1–1:38 pages.

[33] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?. In *annual meeting of the Florida Association of Institutional Research, February.* 1–3.

[34] G. Scanniello, M. Risi, P. Tramontana, and S. Romano. 2017. Fixing Faults in C and Java Source Code: Abbreviated vs. Full-Word Identifier Names. *ACM Trans. Softw. Eng. Methodol.* 26, 2, Article 6 (2017), 43 pages.

[35] S. Shapiro and M. Wilk. 1965. An analysis of variance test for normality. *Biometrika* 52, 3-4 (1965), 591–611.

[36] D. I. K. Sjoberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N. Liborg, and A. C. Rekdal. 2005. A Survey of Controlled Experiments in Software Engineering. *IEEE Trans. on Softw. Eng.* 31, 9 (2005), 733–753.

[37] J. L. Szalma and P. A. Hancock. 2011. Noise effects on human performance: a meta-analytic synthesis. *Psychological bulletin* 137, 4 (2011), 682–707.

[38] S. Vegas, C. Apa, and N. Juristo. 2016. Crossover Designs in Software Engineering Experiments: Benefits and Perils. *IEEE Trans. on Softw. Eng.* 42, 2 (2016), 120–135.

[39] J. Verelst. 2004. The Influence of the Level of Abstraction on the Evolvability of Conceptual Models of Information Systems. In *Procedeedings of the International Symposium on Empirical Software Engineering.* IEEE Computer Society, 17–26.

[40] S. Wellek and M. Blettner. 2012. On the Proper Use of the Crossover Design in Clinical Trials. *Dtsch Arztebl International* 109, 15 (2012), 276–281.

[41] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering.* Springer.

[42] M. Züger, C. Corley, A. N. Meyer, B. Li, T. Fritz, D. Shepherd, V. Augustine, P. Francis, N. Kraft, and W. Snipes. 2017. Reducing Interruptions at Work: A Large-Scale Field Study of FlowLight. In *Proc. of the CHI Conference on Human Factors in Computing Systems.* ACM, 61–72.