A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development

Itir Karac, *Member, IEEE*, Burak Turhan, *Member, IEEE*, and Natalia Juristo, *Member, IEEE*

Abstract—*Background:* Test-Driven Development (TDD) is an iterative software development process characterized by test-code-refactor cycle. TDD recommends that developers work on small and manageable tasks at each iteration. However, the ability to break tasks into small work items effectively is a learned skill that improves with experience. In experimental studies of TDD, the granularity of task descriptions is an overlooked factor. In particular, providing a more granular task description in terms of a set of sub-tasks versus providing a coarser-grained, generic description. *Objective:* We aim to investigate the impact of task description granularity on the outcome of TDD, as implemented by novice developers, with respect to software quality, as measured by functional correctness and functional completeness. *Method:* We conducted a one-factor crossover experiment with 48 graduate students in an academic environment. Each participant applied TDD and implemented two tasks, where one of the tasks was presented using a more granular task description. Resulting artifacts were evaluated with acceptance tests to assess functional correctness and functional completeness. Linear mixed-effects models (LMM) were used for analysis. *Results:* Software quality improved significantly when participants applied TDD using more granular task descriptions. The effect of task description granularity is statistically significant and had a medium to large effect size. Moreover, the task was found to be a significant predictor of software quality which is an interesting result (because two tasks used in the experiment were considered to be of similar complexity). *Conclusion:* For novice TDD practitioners, the outcome of TDD is highly coupled with the ability to break down the task into smaller parts. For researchers, task selection and task description granularity requires more attention in the design of TDD experiments. Task description granularity should be taken into account in secondary studies. Further comparative studies are needed to investigate whether task descriptions affect other development processes similarly.

Index Terms—Test-driven development, programming task description, controlled experiment, empirical software engineering, crossover experiment, software quality, requirement granularity

1 Introduction

Test-Driven Development (TDD) is an iterative software development process characterized by the test-code-refactor cycle (also known as the red-green-refactor cycle) [1]. In each cycle, a micro feature is implemented by following these steps:

- Choose a small feature to implement during the cycle
- Write a unit test for the chosen feature
- Write only the code necessary to pass the newly added test
- Verify that all test, including the added one, pass
- Refactor to improve internal quality

Proponents of TDD claim that this process promotes both internal and external product quality, as well as developer productivity. Since its popularization in the early 2000s, TDD has been the focus of many empirical studies that have investigated its effectiveness [2], [3], [4], [5], [6], [7], [8].

However, the empirical evidence has been mixed regarding the effects of TDD [9], [10], [11], [12], [13], [14]. This may be due in part to the diverse study contexts that have affected the choice and control of factors such as, participant selection (e.g., students or professionals), programming tasks, and the design and execution of the study. Clarifying the impact of such factors is necessary to resolve inconsistent findings and attain a more comprehensive understanding of the TDD process.

Among these factors, the programming tasks employed as experimental objects and the extent they represent real-world software development projects are considered relevant for the external validity of studies [15], [16]. Accordingly, attributes such as complexity, size, and duration have been taken into account to characterize and categorize the tasks [9], [10], [13], [17]. However, how the task is described has been an overlooked aspect. Even when the same task is used, various studies will differ with respect to the granularity of requirements. For example, in [18] and [2], the task was represented, in a finer-grained manner. It was broken up into a list of sub-tasks, each corresponding to a small feature and altogether building up the end-product. In

- *I. Karac is with the M3S Research Unit, University of Oulu, Oulu 90014, Finland. E-mail: itir.karac@oulu.fi.*
- *B. Turhan is with Monash University, Clayton, VIC 3800, Australia. E-mail: burak.turhan@monash.edu.*
- *N. Juristo is with Escuela Tecnica Superior de Ingenieros Informaticos, Universidad Politecnica de Madrid, Madrid 28040, Spain. E-mail: natalia@fi.upm.es.*

Manuscript received 1 June 2018; revised 16 Apr. 2019; accepted 19 May 2019. Date of publication 3 June 2019; date of current version 16 July 2021. (Corresponding author: Itir Karac.)
Recommended for acceptance by E. Murphy-Hill.
Digital Object Identifier no. 10.1109/TSE.2019.2920377

contrast, a coarser-grained representation was used in [19]. The same task was presented, but without an explicit breakdown. The level of granularity in presenting a task is important for TDD, because *choosing a small feature* and maintaining short development cycles are key elements of the process. In fact, the significance of short development cycles is not unique to TDD; it is also recognized in the Agile development paradigm. Alistair Cockburn, a co-author of the *Agile Manifesto* [20], asserts that "Agile developers apply micro-, even nano-incremental development in their work". He demonstrates in his well-known Elephant Carpaccio exercise how developers can break stories into thin, vertical slices with an analogy to slicing an elephant into slices as thin as carpaccio [21]. In the context of TDD, the significance of short development cycles and its contribution to software quality is also supported by empirical evidence [4]. However, novice practitioners are not able to maintain a steady and fast rhythm as well as experts in applying TDD [5].

In light of the significance of this characteristic of the TDD process, providing a task description with more granular requirements may facilitate the developers in focusing on smaller tasks, and as a result to working in shorter cycles. Hence, the granularity of the task description may directly impact the development process and its outcome. An understanding of the interaction between the task description and the TDD process will provide further insight in the dynamics of TDD. Furthermore, the understanding of this interaction is critical from an experimental point of view, as such an interaction poses a major threat to the internal validity of empirical studies in which the development process itself is under investigation.

The goal of this study is to investigate the impact of task description granularity on the outcomes of TDD. A specific outcome of interest is software quality which is assessed on the basis of functional correctness and functional completeness characteristics identified in the ISO/IEC 25010 Product Quality Model [22].

Research Question: Does task description granularity impact functional correctness and functional completeness of the software developed by novice developers using TDD?

The contributions of this study are threefold:

1) Provides a thorough analysis of a controlled experiment in an academic setting to understand the performance of novice TDD developers with respect to task description granularity (A replication package is available online at [23])
2) Demonstrates of the impact of task description granularity on software quality created with TDD
3) Provides empirical evidence on the importance of task selection in TDD experiments

We believe, these contributions will have two major impacts:

1) Calls attention (in practice and education) to the non-triviality of the "choose a small task" step of the TDD process for novice developers
2) Provides insight on factors that impact TDD experiments for consideration when designing future studies

Such insight will be beneficial for practitioners, educators and researchers. Not only is it important to refine our understanding of the dynamics of TDD, but it is also important to facilitate the accurate interpretation, comparison, and aggregation of any empirical findings.

This paper is organized as follows: We introduce related work in Section 2 and describe the experimental setting in Section 3. The results of the data analysis are reported in Section 4, followed by a discussion of the results in Section 5. We assess the threats to the validity of the study in Section 6 and conclude with the possible impact of our results on research and practice along with plans for future study in Section 7.

2 Related Work

In their empirical study on the effectiveness of TDD [2], Erdogmus et al. divided Robert Martin's classic Bowling Score Keeper (BSK) [25] problem into several sub-tasks,[1] coinciding with unique sub-features of the expected end product. Both a control group and experiment group (TDD) were instructed to follow an incremental development approach by implementing and testing sub-tasks, one at a time. However, the control group was asked to first implement a subtask and then write corresponding tests. Whereas the experimental group, conforming to the TDD process, wrote tests before implementing a sub-task. These measures were taken to prevent control group participants from deferring testing, or even omitting completely. In such a case, any comparison of the two development methods would have suffered from a bias due to differences in the testing effort. The authors further discussed that task presentation, when accompanied with corresponding acceptance test methods for each subfeature, facilitated a more accurate assessment of incomplete implementations. Although the effects of this intervention was considered with respect to validity threats, the consideration was limited to a possible bias in productivity due to an increase in testing overhead. However, since task description was not explicitly a factor in the experiment's design, no reliable conclusion about its effect can be inferred.

In a similar study, Munir et al. [18] conducted an experiment to compare TDD and test-last development with respect to internal and external code quality and developer's productivity in which they also used the BSK task. Akin to [2], the task description was described in terms of sub-tasks and corresponding acceptance tests were instrumental in the measurement of the dependent variables. However, the task was decomposed into 7 sub-tasks as opposed to 13 sub-tasks in [2]. The rationale using a different granularity in the task description was not made explicit, nor was this decision's possible effects on the study discussed.

Additional empirical studies on TDD, in which the experimental task was BSK, are summarized in Table 1. The summary shows task description granularity, number and type

1. The subtasks are referred to as user stories in [2], [6], [18], [24] and as sliced requirements in [26]. However, since they are not in a formal user story structure, we prefer using a more generic term.

TABLE 1
Summary of Studies in Which BSK Was Used as the Experimental Task

Study	Task description granularity	Participants	Control Develop. Method	Quality	Productivity
Erdogmus [2]	13 sub-tasks	24 students	Iterative Test-last	no difference	increases
George [19]	no sub-tasks breakdown	12 professional pairs	Waterfall	increases	decreases
Munir [18]	7 sub-tasks	13 professionals	Iterative Test-last	no difference	no difference
Fucci [24]	13 sub-tasks	58 students (22 working in pairs)	Iterative Test-last	no difference	no difference
Tosun [6]	13 sub-tasks	24 professionals	Iterative Test-last	no difference	no difference

of participants, development method for the control group, and the results of the experiment on software quality and developer productivity. Software quality was assessed by black-box acceptance tests and measured as the percentage of passed acceptance tests in all four studies. Productivity was measured as the total development time in [19] and as the percentage of implemented sub-tasks in the other studies. Four of these studies employed task descriptions in terms of sub-tasks, and the conclusions among the studies are mostly consistent. However, the results of the study by George, in which the task description did not contain a breakdown of the task into sub-tasks, diverged from the rest. Of course, it is not possible to attribute the disagreement in the outcomes of these studies specifically to the use of different task descriptions, since the studies varied in a number of other aspects. However, it is important to note that the task description was a facet of the study and did have a potential impact on the outcome.

Development method is not a factor in our study. We investigated the impact of the granularity of the task description on the quality of the software developed using TDD. To the best of our knowledge, this impact has not been investigated directly with empirical methods in the literature. The results of this study will provide insight regarding this overlooked experimental factor in the literature.

Regarding a different line of research on TDD, Fucci et al. investigated the impact of different process characteristics of TDD on software quality and developer productivity [4]. The process characteristics considered in the study were: the order in which tests and production code were written; the average duration of development cycles; that duration's uniformity; and refactoring effort. Their study demonstrated that the duration and the uniformity of cycles, rather than the order in which code and tests were written, contributed to the improvement in software quality and developer productivity. These findings are in accordance with [5] which showed that TDD experts exhibit shorter and more uniform development cycles than novices. If more granular task descriptions promote shorter development cycles, it may reflect this characteristic of experts. The results of these studies constitute a basis for the conjecture that the granularity of the task description may affect the outcome of experiments especially in the context of TDD.

While the aforementioned studies [4], [5] suggest a positive relationship between task representation granularity and the outcomes of TDD, it is theoretically possible that such a representation may cognitively hinder the creative design process of developers [27]. The notion that a problem or situation can be presented in different forms is referred as the *framing* of the concept. The effect of framing on human cognition has been investigated in diverse domains, such as sociology, psychology, marketing science, information systems and software engineering [28], [29]. In software engineering domain, tasks can be framed as "*the system shall...*" requirements [30], a set of user stories [31], or a set of use-case narratives [32]. There are many empirical studies in software engineering investigating framing effects [33], [34], [35], [36]. An experiment reported by Mohanani et al. [37] demonstrated that creativity in conceptual design decreases when the task specification is framed as 'requirements' compared to when it is framed as 'ideas.' Additionally, evidence from the literature suggests that problem structuring reduces design performance [38]. Although creativity and design performance were not within the scope of this study, it is worth noting that granular task descriptions that involve more structured framing of tasks may hinder the development process. This was taken into account when composing our hypotheses.

3 Experimental Setting

In this section, we state our research objectives and describe in detail the experimental setting.

3.1 Research Objectives

The goal of this experiment is to investigate the impact of task description granularity on the outcomes of TDD, specifically on software quality. In accordance with the guidelines for reporting software engineering experiments presented in [39], we have framed our research objectives using the Goal-Question-Metric template suggested by Basili [40]. Our goal is to:

Analyze task descriptions
For the purpose of comparison
With respect to functional correctness and functional completeness of the software developed using TDD
From the point of view of researchers
In the context of an academic course with graduate level students (as proxies for novice developers)

3.2 Design

The factor under investigation is task description, specifically its granularity. It was examined at two levels: the *finer-grained (FG)* task description and the *coarser-grained (CG)* task description (regarded as the control level). The treatment was administered by providing the participant a task description, either *finer-grained* or *coarser-grained*. Further details on the operationalisation and instrumentation of this construct is presented later in the paper. A repeated measurement design is used for its robustness against variation among participants [41]. All the participants received both treatments, they were given a *coarser-grained* task description

TABLE 2
Experiment Design

	Periods	
	Period 1	Period 2
Sequences	Task: BSK	Task: MR
Sequence 1	coarser-grained	finer-grained
Sequence 2	finer-grained	coarser-grained

for one programming task and a *finer-grained* description for a second task. Consequently, the effect of the treatment can be observed for each participant. However, since participants receive the two treatment levels in a specific order, an imposed order may unintentionally have an effect on the outcome. Taking into account such order effect, a crossover design is preferred. A crossover design is a repeated measures design in which participants are randomly assigned to different sequences of the treatments [41]. As there are two levels for the treatment, there are only two possible sequences: CG-FG (Sequence 1) and FG-CG (Sequence 2). Participants were randomly assigned to one of these sequences.

Further details of the experiment design is presented in Table 2. The experiment was conducted in two periods; each period had a duration of 2 hours and was dedicated to the implementation of a single task. The first period is dedicated to the Bowling Score Keeper task and the second period to the Mars Rover (MR) task. These programming tasks are described in more detail in Section 3.5. The allocation of tasks to periods, i.e., the order in which the participants implemented these tasks, was determined randomly. This order was the same for all participants. Accordingly, for the two groups of participants in the experiment, the task sequence was the same but the treatment sequence (i.e., the task description granularity sequence) was different. The rationale and validity of the design is discussed in detail in Section 3.9.

3.3 Variables

The independent variable is *Task Description Granularity (TASK-GRA)* and is operationalised by having the programming task described at different granularity levels. The *coarser-grained* task description is the control level and the *finer-grained* task description, in which the task was described in terms of sub-tasks, is the experimental level.

The software quality response variable is operationalised by using two of the functional suitability characteristics, *functional correctness* and *functional completeness*, identified in the ISO/IEC 25010 Product Quality Model [22].

Functional correctness is defined as the degree to which the functions provide the correct results with the needed degree of precision [22]. It is measured with a metric, *CORRECTNESS*, which captures the degree to which the software provides correct results according to the acceptance test suite.

Functional completeness is defined as the degree to which the set of functions covers all the specified tasks and user objectives [22]. Functional completeness is measured with *COMPLETENESS* metric which indicates the fraction of the implemented features based on the acceptance test results.

The operationalization and instrumentation of these constructs are outlined in Table 3 and presented in detail in Section 3.6.

3.4 Participants and Sampling

We used convenience sampling to recruit participants for the experiment. Participants were graduate students at the University of Oulu who were enrolled in a graduate-level course on software quality and testing during 2015 fall semester, who volunteered to participate in the study.

To characterize participants, we conducted a brief demographics survey in which participants were asked to report their experience in programming in general, Java programming language, Eclipse IDE, unit testing, JUnit, and TDD, using a 4-point ordinal scale (*None*, *Novice*, *Intermediate* and *Expert*). Fig. 1 presents a summary of the survey responses.

Fig. 1 shows that the majority of the participants identified themselves at least as novices, while a few students declared themselves as experts in programming (10 percent), Java (5 percent) and Eclipse IDE (5 percent). A small percentage of the participants reported no experience with Java (2 percent), Eclipse (9 percent), unit testing (22 percent), JUnit (28 percent) and TDD (31 percent). However, this survey was conducted at the very beginning of the semester and before the participants had received 5-weeks (seven hours per week) long training consisting of lectures and hands-on exercises in Java. During the training period, participants attended a total of four one-hour lectures on unit testing and TDD and 15 hours of hands-on exercise sessions (five three-hour long sessions), in which unit testing and test-driven development were demonstrated and practiced. In each exercise session, the topic is first introduced and demonstrated with examples. Then, participants implemented a small programming task using TDD, following the same procedure as the experimental sessions. They were given two hours to implement the task.

Initially, 52 students volunteered to participate in the experiment. These participants were randomly assigned to

TABLE 3
Operationalisation of Constructs and Instrumentation (IV: Independent Variable, DV: Dependent Variable)

Construct		Operationalisation	Instrumentation
Task Description	(IV)	Task Description Granularity	Two sets of specifications: - coarser-grained level task description - finer-grained level task description
Software Quality	(DV)	Functional Correctness	CORRECTNESS metric measured based on acceptance sets
	(DV)	Functional Completeness	COMPLETENESS metric measured based on acceptance sets

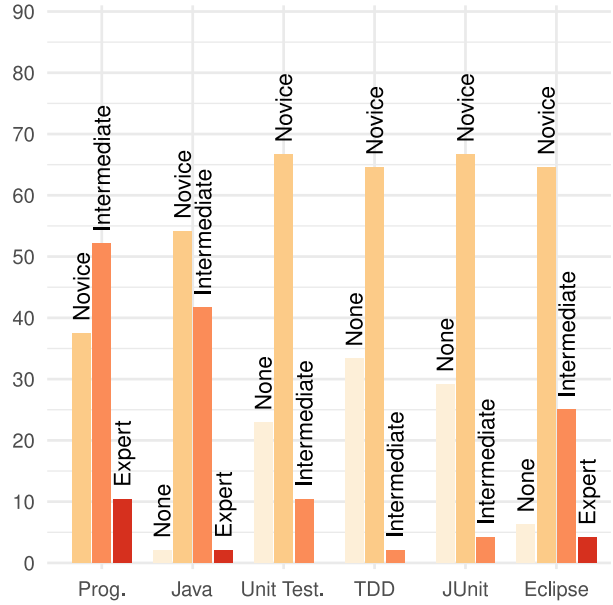

Fig. 1. Summary of participant's experience.

the two groups, corresponding to the two treatment sequences shown in Table 2. Although the initial assignment was balanced, the final group sizes were 23 for Sequence 1 and 25 for Sequence 2, due to attrition.

3.5 Experimental Objects

The tasks used in the experiment were greenfield programming exercises - that require implementation from scratch - and are algorithmic in nature. They were of similar complexity in terms of number of requirements and effort needed for implementation. These exercises are popular in the Agile community and have been previously used in TDD studies [2], [5], [18], [26], [42], [43]. Furthermore, these two tasks have been also used together in experiments on TDD as they are considered to be similar [6], [44].

The first task is Bowling Score Keeper introduced by Robert Martin in [25]. It requires the calculation of the end-game score of a single bowling game for one player. Although the rules are simple, the algorithm is still complicated due to special cases, when the score of a frame depends on the score of the following throws and possible bonus throws at the end of the game.

The second task is called Mars Rover [45], [46]. It requires implementation of a simple application programming interface that handles vertical and horizontal movement of a vehicle on a rectangular grid that wraps around the grid edges and may also contain obstacles. Additionally, some string handling is required for parsing the input and returning the outcome of a run.

Participants were provided task descriptions explaining the requirements of the programming task and also an initial project template for the Eclipse IDE, which included the class and method definitions used in the acceptance test suite. The purpose of these templates was to ensure that the participants complied with the expected interface in the acceptance tests and facilitated the application of the acceptance tests to the participant's code without any adjustment. Participants were instructed that they could extend the structure without changing the existing class and method definitions. The project templates and the acceptance test suits were independent of the treatment and were the same for both the coarser-grained and the finer-grained levels. The project template for the BSK task contained two class definitions and a test class stub, and the template for the MR task contains one stub for the class definition and a second stub for the test class.

3.6 Instrumentation

For the independent variable *Task Description Granularity*, treatments at different levels were administered by using two sets of task description documents, i.e., *coarser-grained* and *finer-grained* descriptions. We employed the task descriptions which were used in previous studies with minor editing [2], [6], [24], [26], [42], [47]. Next, we explain and demonstrate the differences in the task descriptions. The complete task descriptions can be found in the replication package [23].

In the finer-grained task description, the task was decomposed into several small sub-tasks. Each sub-task was presented with a name, a short description, the requirement and an example. Furthermore, they were listed in a specific order, so that each builds upon previous ones, facilitating incremental implementation. Alternatively, coarser-grained level task description presented the task as a whole without any such decomposition and contained one example for the BSK task and two examples for the MR task. In summary, the difference between the finer-grained task description and the coarser grained one was that the former was more structured and more detailed. In summary:

- The subtasks were identified explicitly (13 subtasks for BSK and 11 subtasks for MR)
- Each sub-task was described separately followed by at least one example

Despite these differences in the granularity, both versions of task specifications essentially represent the same functionality, i.e., there were no missing or additional requirements.

The dependent variables *Functional correctness* and *Functional completeness* were measured through acceptance test suites. The acceptance test suites were already available, since the same experimental objects (tasks) were employed in previous work. However, these were not revealed to the participants during the execution of the experiments. After the experiment, the acceptance test suite was applied to the programming artifacts developed by the participants. A summary of acceptance test suits for both tasks in terms of the number of test classes, tests, and assert statements is presented in Table 4. The organization of the tests into test classes was in agreement with the features of the software and tests targeting those features. When the acceptance test was executed, the outcomes for all assert statements in the acceptance test suite were recorded and used in the computation of *CORRECTNESS* and *COMPLETENESS* metrics. *CORRECTNESS* was calculated as the percentage of passing assert statements in the acceptance test suite for the task according to the following formula:

$$CORRECTNESS = \frac{\#Assert(SUCCESS)}{\#Assert(ALL)} \times 100. \quad (1)$$

$\#Assert(SUCCESS)$ denotes the number of successful assert statements when the acceptance test suite is applied to the

TABLE 4
Acceptance Test Suites

Task	Num. of test classes	Num. of acceptance tests	Num. of asserts
MR	11	52	89
BSK	13	57	64

program developed by the participant and $\#Assert(ALL)$ denotes the total number of assert statements in the acceptance test suite. *CORRECTNESS* is measured with a ratio scale in $[0, 100]$

COMPLETENESS was computed as the percentage of the covered features in the program developed by the participant. A feature was considered to be covered if there was at least one passing assert statement in the test class corresponding to that feature. The number of covered features were calculated according to Equation (2).

$$\#Cov.\,Features = \sum_i \begin{cases} 1 & \#\text{ASSERT}_i(\text{SUCCESS}) \geq 1 \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where $\#\text{ASSERT}_i(\text{SUCCESS})$ indicates that the number of successful assert statements in the test class corresponding to the ith feature. Accordingly, *COMPLETENESS* was calculated as shown in Equation (3).

$$COMPLETENESS = \frac{\#Covered\,Features}{\#Features} \times 100, \tag{3}$$

where $\#Features$ indicates the total number of features and it was 13 for the BSK task and 11 for the MR task as shown in Table 4. *COMPLETENESS* was measured with a ratio scale in $[0, 100]$

The allocated time for implementation of the task was fixed at two hours for each task. Therefore, development time was not taken into consideration in the evaluation of *CORRECTNESS* and *COMPLETENESS*.

3.7 Hypotheses

We have formulated our research question in two hypotheses denoted as H_{Cor} and H_{Com} which correspond to *CORRECTNESS* and *COMPLETENESS* respectively. The following formulations specify the null and alternative hypotheses for each:

- H_{Cor0} Task description granularity does not have an effect on CORRECTNESS. (Null Hypothesis)
  H_{Cor1} Task description granularity has an effect on CORRECTNESS.
- H_{Com0} Task description granularity does not have an effect on COMPLETENESS. (Null Hypothesis)
  H_{Com1} Task description granularity has an effect on COMPLETENESS.

Both of the hypotheses are formulated as two-tailed since we don't have theoretical or empirical support to presume a direction for the effect.

In addition to these main hypotheses, we performed additional tests to confirm assumptions related to construct and internal validity of the study, i.e., to check the validity of the assumptions of statistical tests.

3.8 Analysis Approach

We employed linear mixed-effects models (LMMs) for the data analysis. LMMs is the proposed analysis method for crossover experiments in software engineering in [48] as it can handle correlated data resulting from repeated measurements and allows the modeling of variability among participants.

In a typical use of LMMs for analyzing experimental data, a model predicting the dependent variable is formulated in terms of the experimental factors as fixed effects and the participants as random effects [49]. In this study, the treatment *Task Description Granularity*, as the primary focus of investigation, is a fixed effect. Additionally, in accordance with our experimental design presented (see Table 2), *TASK* and *SEQUENCE* are also considered as fixed effects[2] and participants are random effects. An LMM specified in this manner, referred to as a random intercept model, facilitates hypothesis testing for the significance of the fixed effects while accounting for possible correlation between the measurements from the same participant and varying baselines for different participants. In common LMM notation, this model is expressed as

$$\begin{aligned} DV \sim\ & TASK\_GRA + TASK + SEQUENCE \\ & + (1|PARTICIPANT). \end{aligned} \tag{4}$$

Based on this model the statistically significant effects are first determined using Type II Wald tests. Next, the factors that are not significant are dropped and the effect sizes are investigated according to the formulations for crossover design studies given in [50] based on the reduced model.

Additionally, the validity of the fitted models are confirmed by carrying out pos-thoc model diagnostics to check whether the underlying assumptions for the mixed-effects models are satisfied [51]. The normality of the errors are checked with Shapiro-Wilk test and visual inspections of the histogram, normal Q-Q plot, and boxplot of residuals. When there is considerable deviation from normality, transformations (such as, square root, arcsine, logarithmic) of response variables are used. To ensure the independence of the residuals from the factors, fitted values versus residuals plots are examined.

The model fit is evaluated based on goodness of fit measures such as the Akaike's information criterion (AIC), the Schwarz's Bayesian information criterion (BIC), and the log-likelihood. Additionally, the variance explained by the model is evaluated in terms of marginal R^2m which quantifies the variance explained only by the fixed factors and conditional R^2c which quantifies the variance explained by the fixed and random factors [52]. Next, alternative models are considered, which include other fixed factors such as the participant's experience level in programming, Java, unit testing, and TDD. The model fit is then assessed with respect to these alternative models by performing statistical tests based on the log-likelihood ratio.

The analysis was conducted in RStudio environment [53] using the R programming language [54]. LMM computations

2. We are not interested in the main effects of *Task* and *Sequence*, but we introduce them as fixed effects, enforced by the repeated measure design of the experiment, to account for their potential effects in determining the main effect of *Task Description Granularity*.

were carried out using the nlme package for R [55]. For effect size analysis, the emmeans package[3] and R package provided with [50] was utilized. The MuMIn package[4] was used to compute the variance explained by the model.

3.9 Evaluation of Design Validity

A crossover design was suitable for our purposes, as it requires fewer participants compared to parallel designs to achieve similar power. Second, since each participant received the treatment at both levels, the effect of the treatment was a within-subject factor and was less sensitive to variation among participants [48]. Furthermore, this design allowed the investigation of any effects that may have been caused by the order of the treatments received or carryover. Even though such effects were not expected in this study, the chosen design allowed this exploration.

The validity of a crossover design may be exposed to threats due to carryover, fatigue, order, and practice effects. Since the experiment was conducted in the context of a graduate-level course, students participated in five weekly hands-on exercise sessions prior to the experimental sessions and the experiment was conducted under the same conditions fatigue or practice were not likely to have had a major effect.

Regarding carryover or an order effect, the difference between the levels of treatment was the granularity of task descriptions. Since our participants were students, they were accustomed to implementing programming assignments based on task descriptions provided to them. It is unlikely, that their performance in the second period would be unduly influenced by the previous task description condition. We investigated the order in the analysis to verify this presumption.

Furthermore, in order to evaluate the validity of the findings with respect to other threats to validity, we conducted surveys immediately after each experimental session. In these surveys, participants were asked to evaluate the difficulty of the tasks and the adequacy and comprehensibility of the task descriptions. The responses were gathered with a 5-point Likert scale. Additionally, participants assessed how much they complied with the TDD process on a scale of 0 to 100. The results of this evaluation are presented in Sections 5 and 6.

3.10 Experiment Execution

The experiment was conducted within the context of a graduate-level course in University of Oulu over eight weeks. Before the experiment, we prepared the development environment, experimental objects, and data collection tools and trained participants on TDD and related concepts. The development environment, Eclipse IDE (Release Luna) and JUnit (Version 4), was prepared on the computers in the university computer labs. The same environment was used for both training and experimental sessions. Additionally, task descriptions, project templates for Eclipse, and GitHub repositories for programming tasks were prepared. For data collection, survey forms for demographic information, task evaluation, and self-assessment of performance were prepared on Google Docs. Also, small scripts were prepared to retrieve the source code and related log files from the GitHub repository of each participant.

The experiment was conducted following the training, in two periods that were one week apart. Briefly, the protocol for each experiment session was as follows:

- Participants obtained project template provided for the task from the GitHub repository
- Task descriptions were handed out to participants
- The task was implemented (two hours)
- Participants submitted their implementation to Github
- Survey for task and performance assessment were completed by participants
- Participants' code and related files were retrieved from their Github repositories and saved to local storage
- Participants' responses to the survey were downloaded from Google Docs

Following the experimental sessions, acceptance test suites were applied to the programs developed by the participants and *CORRECTNESS* and *COMPLETENESS* metrics were computed as described in Section 3.6.

The experimental package is available online for other researchers to replicate our experiment [23].

4 Results

In this section, we report the results of the data analysis. For each response variable, we present the descriptive statistics and the plots depicting data, followed by the results of the statistical data analysis conducted using linear mixed-effects models, and the discussion on model validity and fit.

4.1 Functional Correctness

4.1.1 Descriptive Statistics

Table 5 presents the descriptive statistics for *CORRECTNESS* at *coarser-grained* and *finer-grained* levels of *Task Description Granularity*. The mean value for *CORRECTNESS* at the *finer-grained* level was 47.7 with a 95 percent confidence interval of $[38.26, 57.14]$. At the *coarser-grained* level, both the mean value (31.64) and limits of 95 percent confidence interval ($[24, 39.28]$) were much smaller compared to the *finer-grained* level. The trimmed mean values are in accordance with mean values. Dispersion at both levels were considerably large compared to the means. Additionally, the wide range, 91.0 for *coarser-grained* and 100.0 *finer-grained*, indicated a great amount of variability among the participants. This is also apparent in violin plots presented Fig. 2. The range and the interquartile range was larger for the *finer-grained* level. Skewness and excess kurtosis measures indicated approximately symmetric but flatter distributions for the *finer-grained* level and moderately right-skewed but a more normal-like distribution for the *coarser-grained level*.

In Fig. 3, the boxplots for *CORRECTNESS* at both levels of *Task Description Granularity* are shown separately for different tasks (a) and for different sequences (b). Fig. 3a reveals that the improvement in *CORRECTNESS* for the *finer-grained* task description was consistent across both tasks, but the magnitude of the improvement was considerably larger for

3. https://cran.r-project.org/web/packages/emmeans
4. https://cran.r-project.org/web/packages/MuMIn/

TABLE 5
Descriptive Statistics for CORRECTNESS

statistic	coarser-grained	finer-grained
Mean (Std. Err.)	31.64 (3.8)	47.70 (4.69)
95 % CI for mean	(24, 39.28)	(38.26, 57.14)
5% Trimmed Mean	30.54	47.50
Median	23.44	42.97
Std. Deviation	26.33	32.47
Min.- Max.	0 - 90.62	0 - 100.00
Range	90.62	100.00
Interquartile Range	31.23	56.25
Skewness	0.85	0.24
Kurtosis	-0.36	-1.26

the MR task. Another observation is that the mean and the variance of *CORRECTNESS* are much smaller for the MR task. It should be noted that the comparison of *CORRECTNESS* at the *coarser-grained* and *finer-grained* levels for a given task was a between-subject comparison since the participants implemented each task only once. Nevertheless, these observations suggest that task may have had an impact on *CORRECTNESS*. Similarly, Fig. 3b presents the boxplots for *CORRECTNESS* at *coarser-grained* and *finer-grained* levels separately for Sequence 1 and Sequence 2. Sequence 1 was the group of participants who first implemented the BSK task using the *coarser-grained* task description, and then the MR task using the *finer-grained* task description. Sequence 2 was the group of participants who implemented the BSK task first, but using the *finer-grained* task description, and then the MR task using the *coarser-grained* task description. The plot indicates an interaction between the sequence and the relationship between *Task Description Granularity* and *CORRECTNESS*. For Sequence 2, the mean *CORRECTNESS* improved with the *finer-grained* task description. *CORRECTNESS* at the *finer-grained* level had a smaller mean but a larger median value for Sequence 1. However, it should be noted that the comparison of *CORRECTNESS* for different *Task Description Granularity* levels within a sequence was a between-subject, between-task comparison. Hence, the complex interaction may be due to the variation among the participants and tasks.

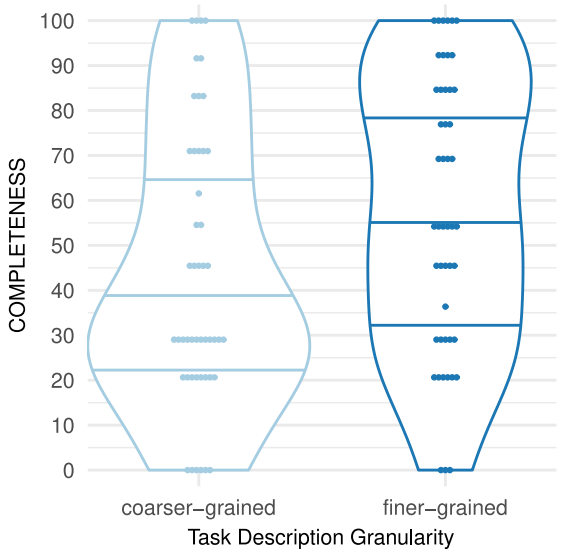

Fig. 2. Violin plots for *CORRECTNESS* at both levels of *Task Description Granularity*. Data points are indicated as dots grouped into bins of width 5. Horizontal lines within the plots indicate 0.25, 0.50, and 0.75 quantiles.

4.1.2 Linear Mixed-Effects Model

The statistical analysis of *CORRECTNESS* was conducted using an LMM as outlined in Section 3.8. We first present the model, the results of the statistical tests for effects, and the model diagnostics. To ensure that the assumptions of the LMM are satisfied, a square root transformation was applied on *CORRECTNESS*.

The final LMM for *CORRECTNESS* (Formula (5)) included terms for *Task Description Granularity* and *Task* as fixed factors and *Participant* as a random factor. The *Sequence* term was dropped from the base model given in Formula (4) because it was found to be not significant ($F_{1,46} = .35$, $p = .57$)

$$sqrt(CORRECTNESS) \sim TASK\_GRA + TASK + (1|PARTICIPANT). \quad (5)$$

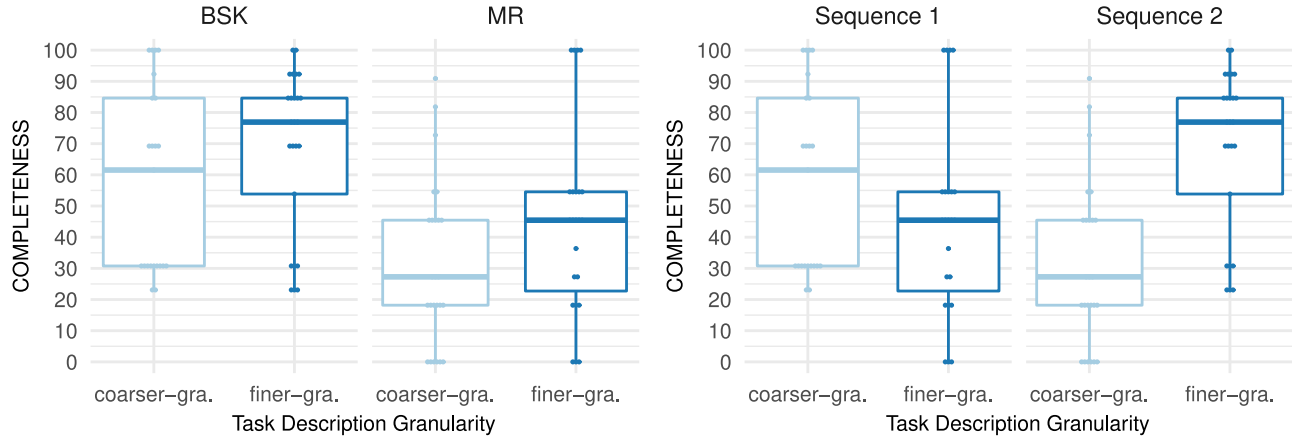

Fig. 3. Boxplots for *CORRECTNESS* shown separately for different tasks and sequences. Data points are indicated as dots grouped into bins of width 5. Horizontal lines within the plots indicate 0.25, 0.50, and 0.75 quantiles.

TABLE 6
Estimates of Fixed Effects for *CORRECTNESS*

Parameter	Estimate	Std. Err.	95% Conf. Interval
Intercept	6.26	0.40	(5.47, 7.05)
TASK_GRA:FG	1.25	0.35	(0.55, 1.95)
TASK:MR	-2.49	0.35	(-3.19, -1.79)

The parameter estimates for fixed effects and the confidence intervals are given in Table 6. The parameter *TASK_GRA:FG* denotes that the coefficient corresponds to the *finer-grained* level, i.e., the *coarser-grained* level was taken as the baseline. In the same manner, *TASK:MR* indicates that the BSK task was considered as the baseline. The interpretation of these estimates should be done carefully, keeping in mind that they are computed based on transformed values of *CORRECTNESS*. For *TASK_GRA*, the estimate 1.25 indicates that *CORRECTNESS* for the *finer-grained* task descriptions was higher compared to the *coarser-grained* task descriptions. But due to the data transformation, the magnitude of the difference cannot be inferred accurately based on the value of the parameter. Nonetheless, the confidence interval $(0.55, 1.95)$ which contains only positive values, verifies the direction of the effect, namely an increase in *CORRECTNESS* for *finer-grained* task descriptions. For the second fixed factor *TASK*, the estimate -2.49 indicates that *CORRECTNESS* was higher for the BSK task, which is in accordance with the plots in Fig. 3a.

4.1.3 Hypothesis Tests

The significance of the effects of *Task Description Granularity* and *Task* in this model were investigated using the Wald test and the results are presented in Table 7. The null hypothesis that the coefficient of the term corresponding to a predictor is zero in the model was rejected for both factors at a significance level ($p < .01$). Hence, *Task Description Granularity* and *Task* had significant effects on *CORRECTNESS* and $H_{Cor\,0}$ is rejected.

4.1.4 Effect Size

The effect size of *Task Description Granularity* was investigated first by evaluating the Estimated Marginal Means (also known as the Least Square Means) after transforming the fitted values back to the original scale. The results presented in Fig. 4 reveal that the predicted average increase in *CORRECTNESS* with *finer-grained* task descriptions was similar for both tasks. The estimated marginal means of *CORRECTNESS* for *coarser-grained* and *finer-grained* task descriptions are 25.13 $(SE = 3.56)$ and 39.24 $(SE = 4.45)$ respectively, indicating a 56.15 percent increase. Additionally, two standardized effect size measures for crossover design were computed according to the formulation given in [50]. The

TABLE 7
Tests of Fixed Effects for *CORRECTNESS*

Source	Numerator df	Denominator df	F	Sig.
TASK_GRA	1	46	12.98	.001
TASK	1	46	51.49	$< .001$

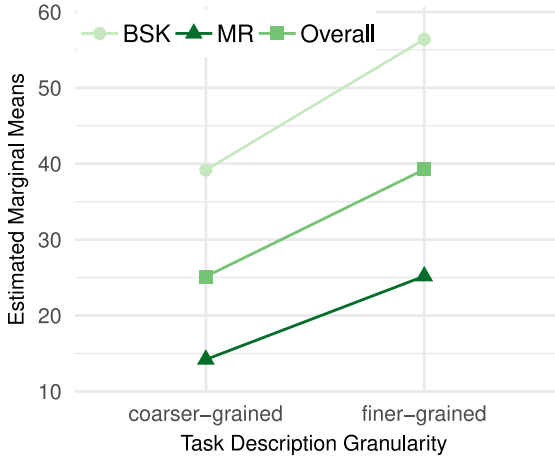

Fig. 4. Profile plots of estimated marginal means for CORRECTNESS.

standardized effect size comparable to the independent groups design ($g_{IG} = 0.50$) and standardized effect size comparable to the repeated measures design ($g_{RM} = 0.71$) indicate a medium to large effect.

4.1.5 Model Validity and Fit

The validity of the model was established by first constructing a model aligned with the experiment design and then dropping the factors that were not significant as described in Section 3.8. Second, post-hoc model diagnostics were carried out to check whether the underlying assumptions for the mixed-effects models were valid for the final fitted model. Finally, the model fit was evaluated.

Diagnostics for the model specified in Formula (5) was carried out by inspecting residuals of the model. An approximately symmetric distribution around zero (M = 1.53×10^{-16}) and a linear trend in the normal Q-Q plot of residuals indicated that the residuals were normally distributed which was also verified by the Shapiro-Wilk test. The null hypothesis that the residuals follow a normal distribution cannot be rejected ($W = 0.99$, $p = .16$). The predicted values versus residuals plot did not indicate any major deviations from a linear form and also showed relatively constant variance across the fitted range. The diagonal line of points was most likely caused by the the small cluster of zeros in the data. The slight increase in variance to the right of the center was likely a result of having fewer observations in these predicted areas. Hence, the observed variance doesn't necessarily indicate an issue with the specification of the model.

The model fit was assessed as explained in Section 3.8. The alternative models were built using forward selection starting from the model with only the random factor. The factors considered were the experimental factors, i.e., *TASK_GRA*, *TASK*, and *SEQUENCE*, followed by the control factors related to the participants' experience in programming (*EXP_PROG*), the Java programming language (*EXP_JAVA*), unit testing (*EXP_UT*) and TDD (*EXP_TDD*). In addition to the fixed factors, all of the models included participants as a random factor to reflect the repeated measures design. These models are listed in Table 8 along with model fit statistics, i.e., conditional and marginal R^2, the Akaike's information criterion, the Schwarz's Bayesian information criterion and log-likelihood (LL) statistic. Model 3 in this table corresponds

TABLE 8
Comparison of Alternative Models for *CORRECTNESS*

Model for *CORRECTNESS*		Variance explained		Model fit			Comparison of models			
No	Fixed factors	R^2m	R^2c	AIC	BIC	LL	Test	df	LL ratio	p-value
1	1	0.00 %	12.4 %	475.84	483.53	-234.92		3		
2	TASK_GRA	5.88 %	24.09 %	471.00	481.26	-231.50	1 vs 2	4	6.84	$< .001$
3	TASK_GRA + TASK	25.68 %	63.45 %	438.01	450.83	-214.01	2 vs 3	5	34.99	$< .001$
4	TASK_GRA + TASK + SEQUENCE	26.08 %	63.45 %	439.66	455.05	-213.83	3 vs 4	6	0.35	.555
5	TASK_GRA + TASK + EXP_PROG	27.86 %	63.46 %	438.12	453.50	-213.06	3 vs 5	6	1.90	.169
6	TASK_GRA + TASK + EXP_JAVA	32.15 %	63.48 %	434.16	449.55	-211.08	3 vs 6	6	5.85	.016
7	TASK_GRA + TASK + EXP_JAVA + EXP_UT	32.15 %	63.48 %	436.16	454.11	-211.08	6 vs 7	7	0.00	.978
8	TASK_GRA + TASK + EXP_JAVA + EXP_TDD	32.18 %	63.48 %	436.14	454.09	-211.07	1 vs 6	8	0.02	.875

to the LMM described in Formula 5. The fraction of total variance explained by this model was 63.45 percent, where the fixed factors *TASK_GRA* and *TASK* accounted for 25.68 percent of the total variance. The subsequent models exhibited a slightly larger R^2m but very similar R^2c values, which suggests that the additional control factors in these models explain the part of the variance that was already accounted for in Model 3 by the *Participant* random factor. The largest increase in R^2m was attained by Model 6, which includes also the participant's Java experience level. These observations were verified by statistical tests performed for pairs of nested models based on the log-likelihood ratio. The results of these tests, reported in Table 8 under a comparisons of models header, confirm that Model 3 and Model 6 were significantly better ($p < .05$) than the models nested within them. Nevertheless, the results of hypothesis tests about fixed factors and effect sizes reported in previous subsections hold because the parameter estimates for *TASK_GRA* and *TASK* with Model 6 are the same as Model 3 (up to three significant digits).

4.2 Functional Completeness

4.2.1 Descriptive Statistics

Descriptive statistics for *COMPLETENESS* are presented in Table 9. The mean *COMPLETENESS* value for *coarser-grained* level of *Task Description Granularity* ($M = 43.66$) was smaller than the *finer-grained* level ($M = 57.52$). The trimmed mean values were close to mean values for both levels whereas the median was considerably lower than the mean value at the *coarser-grained* level. However, this seemingly large difference between the mean and the median of the *COMPLETENESS* metric corresponded to a difference of only one unit in the measurement of the number of covered features. Hence, it did not necessarily suggest asymmetry and the skewness value (0.44) did not indicate extreme asymmetry. The distributions for the two levels were similar with respect to dispersion and were both flatter compared to a normal distribution. Although the skewness was in opposite directions, the deviation from symmetry was not large. Fig. 5 shows the violin plots for *COMPLETENESS* at each *Task Description Granularity* level, which can be seen in further detail for tasks and sequences separately in Fig. 6. Observations for *COMPLETENESS* were in agreement with the observations made for *CORRECTNESS*; the improvements in *COMPLETENESS* with finer-grained task description was consistent across tasks, whereas for different sequences, the effect was in opposite directions.

TABLE 9
Descriptive Statistics for COMPLETENESS

statistic	coarser-grained	finer-grained
Mean (Std. Err.)	43.66 (4.45)	57.52 (4.43)
95 % CI for mean	(34.71, 52.61)	(48.61, 66.43)
5% Trimmed Mean	43.09	58.20
Median	30.77	54.55
Std. Deviation	30.81	30.72
Min.- Max.	0 - 100	0 - 100
Range	100	100
Interquartile Range	47.38	53.85
Skewness	0.41	-0.18
Kurtosis	-1.00	-1.21

4.2.2 Linear Mixed-Effects Model

The statistical analysis of *COMPLETENESS* was conducted in the same manner as *CORRECTNESS* while using the arcsine transformation to ensure that the assumptions of LMM were satisfied.

The final *COMPLETENESS* model (Formula (6)) included terms for *Task Description Granularity* and *Task* as fixed factors and *Participant* as a random factor. The *Sequence* term was dropped from the base model given in Formula 4 because it was found not to be significant ($F_{1,46} = .43, p = .52$)

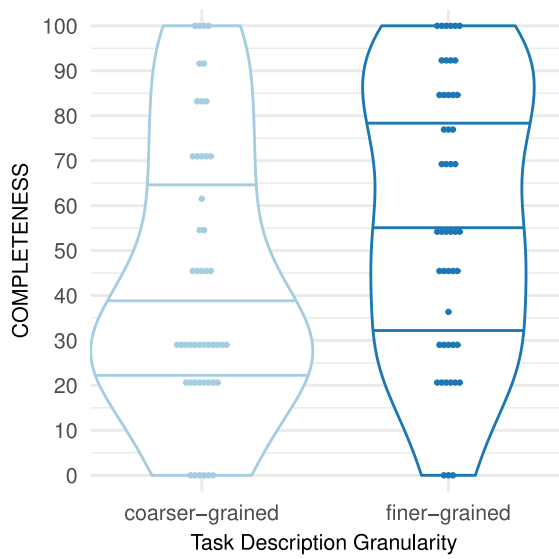

Fig. 5. Violin plots for *COMPLETENESS* at both levels of *Task Description Granularity*. Data points are indicated as dots grouped into bins of width 5. Horizontal lines within the plots indicate 0.25, 0.50, and 0.75 quantiles.

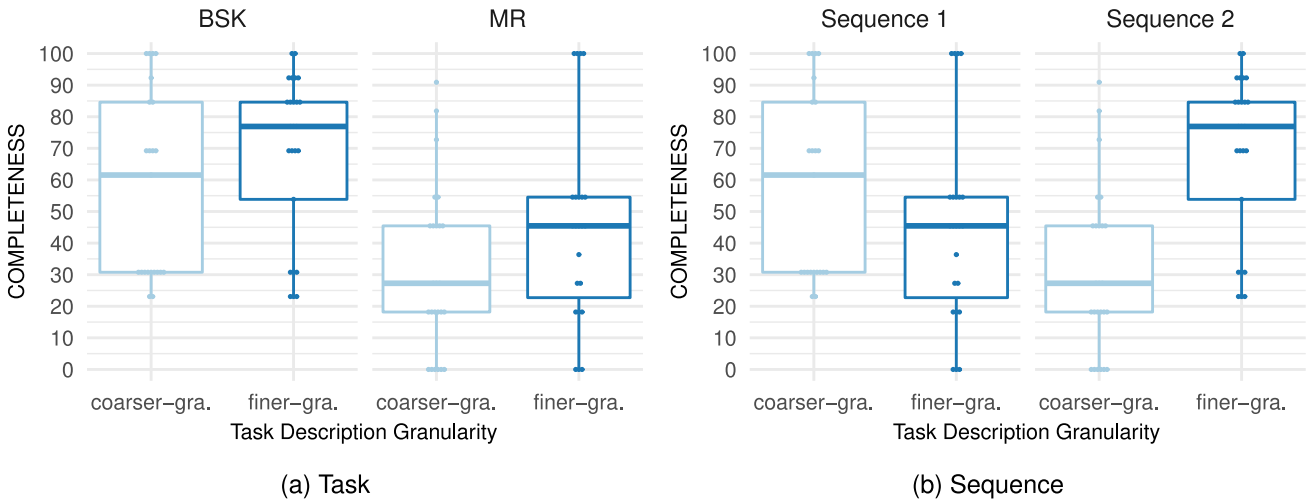

Fig. 6. Boxplots for *COMPLETENESS* shown separately on different tasks and sequences. Horizontal lines within the plots indicate 0.25, 0.50, and 0.75 quantiles.

$$arcsin(COMPLETENESS) \sim TASK\_GRA + TASK + (1|PARTICIPANT). \quad (6)$$

The estimates for these coefficients and confidence intervals are given in Table 10, which indicated an increase in *COMPLETENESS* with the *finer-grained* task description. Additionally, *COMPLETENESS* is lower for the MR task compared to the BSK task, which is also in accordance with the findings for *CORRECTNESS*.

4.2.3 Hypothesis Tests

Results of the Wald tests for fixed effects are presented in Table 11. As for the case of *CORRECTNESS*, both *Task* and *Task Description Granularity* had significant effects ($p < .01$). Hence, $H_{Com\,0}$ is rejected at ($p = .004$).

4.2.4 Effect Size

The Estimated Marginal Means of *COMPLETENESS* for *coarser-grained* and *finer-grained* task descriptions were 42.98 and 59.40 respectively. The increase in *COMPLETENESS* was 38.20 percent which was smaller compared to the increase in *CORRECTNESS*. Fig. 7 depicts the profile plots of estimated marginal means separately for the BSK task and the MR task, revealing that the size of the effect was similar for both tasks. The standardized effect size comparable to independent groups design, ($g_{IG} = 0.43$), suggested a close to medium size effect. Standardized effect size comparable to repeated measures design was ($g_{RM} = 0.57$).

4.2.5 Model Validity and Fit

To verify the validity of the LMM method, the normality of the residuals for *COMPLETENESS* was tested with the Shapiro-Wilk test. As this condition was not met for *COMPLETENESS*, the data was transformed using an arcsine transformation. For the transformed data, the null hypothesis of the Shapiro-Wilk test that the residuals follow a normal distribution was rejected ($W = 0.98, p = .002$). Visual inspection of the histogram, the normal Q-Q plot, and the boxplot of residuals did not indicate a major deviation from normality. The mean residual value was 1.53×10^{-16}, the normal Q-Q plot followed a linear trend except for a slight deviation at the endpoints, which are explained by the bounded nature of the response variable in $[0, 100]$. The deviation did not likely affect the validity, since inference for the fixed effects under the assumption of independent normally distributed errors with constant variance have been shown to be robust when the errors are either non-Gaussian or heteroscedastic [56]. The assessment of model fit was carried out in the same manner as described for *CORRECTNESS* and the results, summarized in Table 12, were in agreement with the findings for *CORRECTNESS*. Specifically, the trend for the variance explained was similar, the increase in R^2m was largest among the control factors when *EXP_JAVA* was included, and Model 3 and Model 6 were significantly better than the models nested within them according to the log-likelihood tests. Consequently, the conclusion that the additional control factors explained the part of the variance that was already accounted for in Model 3 by the *Participant* random factor also holds for *COMPLETENESS*. Furthermore, the parameter estimates for *TASK_GRA* and *TASK* with Model 6 were the same as Model 3 (up to three significant digits). Hence, the results of

TABLE 10
Estimates of Fixed Effects for *COMPLETENESS*

Parameter	Estimate	Std. Err.	95% Conf. Interval
Intercept	1.76	0.13	(1.52, 2.01)
TASK_GRA:FG	0.33	0.11	(0.11, 0.54)
TASK:MR	-0.66	0.11	(-0.88, -0.44)

TABLE 11
Tests of Fixed Effects for *COMPLETENESS*

Source	Numerator df	Denominator df	F	Sig.
TASK_GRA	1	46	9.04	.004
TASK	1	46	37.03	$< .001$

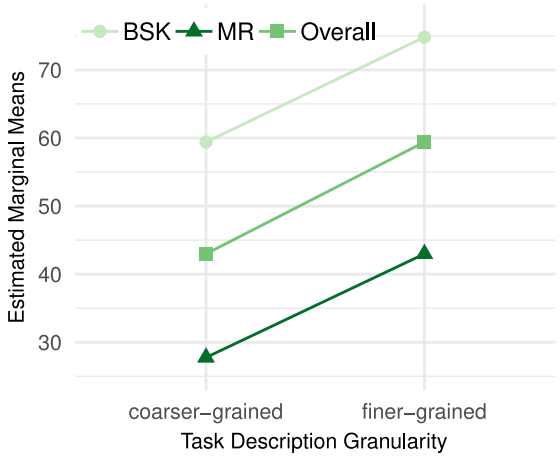

Fig. 7. Profile plots of estimated marginal means for COMPLETENESS.

hypothesis tests about fixed factors and effect sizes reported in previous subsections based on Model 3 remain valid.

5 Discussions

Both the task description granularity and the task had significant impacts on software quality in the use of TDD (i.e., their tests resulted in significant effects with medium to large effect sizes). At first glance, these results seem intuitive. First, the significance of task description granularity is informative for both TDD practitioners and experimenters, as it provides insight into the dynamics of this method and highlights an over-looked factor in experiments, i.e., choosing small tasks to work on. Second, the impact of the task was interesting as the tasks were of similar difficulty in this study. Hence, the reasons of observed differences warrants further attention.

The primary factor investigated in this study was the task description granularity rather than the task itself. Employment of two different tasks was necessary due to the repeated measurement design of the experiment. These tasks, the Bowling Score Keeper and the Mars Rover API, were selected from previous empirical studies on TDD, in which they were considered to be of similar difficulty [4], [6], [26]. Therefore, the observed effect of the task on the outcome was unexpected and needs further consideration. It should be noted that participants delivered higher quality software in the first experimental period for which the Bowling Score Keeper was the task that was randomly chosen. Thus, the learning effect didn't account for the observed difference and other plausible explanations were explored. First, the comparability of these tasks from the participants' perspective was explored based on the responses to a self-assessment questionnaire conducted after each experimental period. In these questionnaires, participants were asked to evaluate the difficulty of the tasks and if the task description was comprehensible and adequate. The responses were measured with a five-point Likert scale and compared using the Mann-Whitney-Wilcoxon test. No statistically significant (at the .05 level) difference in difficulty of the task or comprehensibility and adequacy of the task description was found between the tasks. This conclusion didn't change when the comparison was done separately for *coarser-grained* and *finer-grained* task descriptions. Thus, these two tasks were considered to be of similar difficulty, not only by researchers, but also by participants.

We also considered process conformance to TDD. One possible explanation was that the participants were able to apply TDD better on the BSK task and attain higher software quality. To test this hypothesis, we analyzed the responses in the post-task self-assessment questionnaire that asked how much they were able to comply to TDD process. No statistically significant (at .05 level) difference was detected between the responses for the two tasks. As this analysis was based on the participants' subjective assessment on their conformance to TDD, the conclusion needs to be validated using objective process conformance measures (for further mitigation actions we took on this issue, please see Section 6, threats to internal validity). As Fucci et al. [4] states, the TDD process is characterized not only by the order in which tests are written but also by the granularity and uniformity of the micro-cycles. Furthermore, the investigation of the participants' compliance to TDD based on these dimensions may shed light on the difference between the two experimental tasks. These results not only provide empirical evidence to support that task selection is important in software engineering experiments, but also indicate characteristics other than the task difficulty may effect the results of the experiments and need to be taken into account beyond the external validity of the findings.

To elaborate more on the task description granularity factor, let us revisit the most distinctive characteristics of TDD which are: 1) writing tests before the production code; 2) iterative development with very small development cycles and 3) a great deal of refactoring. Among these, the unconventional "write tests before production code" principle of TDD leads to the perception that TDD is equivalent to test-first development and downplays the importance of the

TABLE 12
Comparison of Alternative Models for *COMPLETENESS*

Model for *COMPLETENESS*		Variance explained		Model fit			Comparison of models			
No	Fixed factors	R^2m	R^2c	AIC	BIC	LL	Test	df	LL ratio	p-value
1	1	0.00 %	22.14 %	244.30	251.99	-119.15		3		
2	TASK_GRA	4.38 %	30.84 %	240.64	250.89	-116.32	1 vs 2	4	5.67	.017
3	TASK_GRA + TASK	19.56 %	61.02 %	215.19	228.01	-102.60	2 vs 3	5	27.44	< .001
4	TASK_GRA + TASK + SEQUENCE	20.11 %	61.02 %	216.76	232.15	-102.38	3 vs 4	6	0.43	.510
5	TASK_GRA + TASK + EXP_PROG	22.19 %	61.03 %	215.09	230.48	-101.55	1 vs 2	6	2.10	.150
6	TASK_GRA + TASK + EXP_JAVA	29.18 %	61.06 %	209.02	224.40	-98.51	1 vs 2	6	8.17	.004
7	TASK_GRA + TASK + EXP_JAVA + EXP_UT	29.19 %	61.06 %	211.01	228.96	-98.50	2 vs 3	7	0.01	.908
8	TASK_GRA + TASK + EXP_JAVA + EXP_TDD	29.18 %	61.06 %	211.01	228.96	-98.51	1 vs 2	7	0.01	.940

micro-cyclic nature of this development technique. However, this feature assures that the focus is on one simple, well-defined sub-task at a time. In order to comply with the TDD process completely, developers need to decompose the task into smaller sub-tasks. This doesn't necessarily mean that a decomposition must be readily on hand before starting the implementation. But, at each iteration, the developer needs to choose the next small sub-task to implement, based on his/her problem solving skills. Hence, the fact that participants achieve higher software quality with more granular task descriptions suggest that breaking down the task into smaller work items is not straightforward for novice developers. In a sense, TDD assumes the possession of this skill by developers, rather than facilitating it.

Young professionals or software engineering students may not have acquired the proper skills and experience to identify small enough subtasks in an iterative manner. As supported by our findings, they attain better software quality with TDD when the task description enables them to identify and choose subtasks for each development cycle. For experienced professionals, writing tests before production code may be instrumental in focusing them on one small feature at a time and working in small development cycles. However, for novice developers, the ability to choose a small tasks is likely to be more dominant in determining the outcome of TDD.

In the context of TDD experiments, our findings raise serious concerns. First, manipulating the granularity of the task description poses a threat to the internal validity of the experiments since task description was shown to influence the observed outcome. Second, in the secondary studies not only the characteristics of the task, but the granularity of the task descriptions needs to be taken into account for accurate aggregation of the findings of primary studies. While finer grained task descriptions fulfill the "choose a simple task" step of TDD, experiment participants exposed to coarser grained task descriptions had to break down bigger tasks ito smaller ones on their own, and the results were likely affected by their ability to do so.

6 Threats to Validity

In this section we discuss potential threats to validity relevant to our study, based on the classification provided in Wohlin et al. [16].

6.1 Construct Validity

The constructs used for the dependent variables in the study and the corresponding measures were well-defined and used in previous studies. Still, our experiment may suffer from mono-method bias since each construct was defined in terms of a single measure. To mitigate this threat, the measures used in previous studies were chosen. The study did not suffer from mono-operation bias since two separate tasks, BSK and MR, were used.

Design threats to construct validity are discussed separately in Section 3.9.

6.2 Internal Validity

The manipulation of task description granularity was performed by providing separate task descriptions for the different levels of granularity. In order to confirm that this had not caused any confounding factors such as the lack of either comprehension of the task or the adequacy of the specifications for any group, we conducted a survey immediately after the task was completed. We asked the participants to evaluate the task specifications in terms of adequacy and comprehensibility. The responses were gathered with a 5-point Likert scale. For both of the attributes, we observed that the median values for coarser-grained and finer-grained task descriptions were the same. The results remain unchanged when the median values were computed separately for each task.

Since the scope of this study was TDD, the participants' ability to follow TDD process was also relevant for the validity of the findings. As a precaution, we provided training on TDD and related concepts before conducting the experiment. Furthermore, we evaluated TDD conformance based on two independent assessments, participant's self assessment obtained with the post-task survey, and assessment with the Besouro Tool [57] using the TDD conformance metric described in [4], [6]. Furthermore, we repeated the statistical analysis after removing data points that did not meet the minimum criteria for TDD conformance which was set at 25 percent. Based on this criteria and the two assessments methods, we investigated two selection schemes. In the first scheme, data points which met the minimum conformance criteria for at least one of the assessments, i.e., self-assessment or assessment with Besouro, were selected. In the second and more strict selection scheme, a data point was selected only if both of the assessments exceeded the minimum criteria. When we repeated the statistical analysis with these two selection schemes, we obtained the same results for the hypothesis tests. Hence, our conclusions were not sensitive to the data points with low TDD conformance.

6.3 External Validity

The experiment was conducted in an academic setting in the context of a graduate level course, limiting the ability to generalize the results to industrial practice. This was partly by design, since our hypothesis focused on novice developers.

The tasks used in the experiment were simple tasks with a two hour implementation duration. Hence, the generalization to more complex real-life tasks is limited. However, if even with simple tasks, the task description granularity has an effect on the outcome of TDD, it is reasonable to expect that it will also have an effect on more complex tasks.

Even though the tasks chosen for the experiment were seemingly similar in complexity, the effect of the task was significant for both of the dependent variables. The fact that the effect of treatment was the same for both tasks suggests that the results can be generalized to tasks with similar complexities.

6.4 Conclusion Validity

The analysis was carried out with LMMs, following the instructions on how to analyze crossover experiments in [48]. The underlying distributional assumptions of LMMs are that the errors are independent of the random effects and follow a normal distribution with mean zero and a constant variance across different levels of the fixed effects. It should be noted

that the the objective of using LMMs to analyze experimental data is not to construct a predictive model, but to test the hypotheses about fixed effects. In this case, the random effects were only included to obtain reliable inference on the fixed effects by taking within-subject correlation into account.

The detailed model diagnostics and validity of the models, discussed for each response variable separately in the results section, do not indicate a severe violation of the assumptions. Moreover, it has been shown that the estimates of the fixed effects are robust to violations of the above-mentioned distributional assumptions, except when the error variance depends on a covariate included in the model that interacts with time [56]. To verify this, the existence of an interaction between *Task* and *Task Description Granularity* was checked formally with statistical tests and rejected ($F_{1,45} = .35, p = .56$ for *CORRECTNESS* and $F_{1,45} = .43, p = .52$ for *COMPLETENESS*).

Additionally, we checked results for LMMs on response variables without any transformations for which the normality assumption was clearly violated. The conclusion of the hypothesis tests did not change for the models which confirms the robustness of the fixed estimates against violations of the normality assumptions. We concluded that the results obtained based on the models specified with Formulas (5) and (6) were valid.

7 Conclusions

In this paper, we investigated an overlooked aspect of TDD, which states that developers need to work on *small* and *manageable* tasks at each iteration. This is an assumption that is usually taken for granted. However, the ability of a developer to break tasks into small work items is a skill that is developed with experience. Hence, we hypothesized that the level of detail (i.e., finer-grained versus coarser-grained) in the provided specifications could have an effect on the quality of the produced code, i.e., functional correctness and functional correctness. Moreover, theory from other disciplines support that this is an issue worth investigating (as discussed in Section 2).

We conducted a controlled experiment with graduate students as proxies for novice developers and detected significant effects for the task description granularity and the task itself, on the response variables. Based on our results, we derived the following conclusions that are relevant for both practitioners and researchers:

- The ability to have problem-solving and divide-and-conquer skills to break specifications into smaller work items is an important practical skill, at least in the context of TDD.
- The analysis and interpretation of the results of experiments should account for the potential impact of the experimental tasks used.

For practitioners, specifically for novices, we emphasize that it is an essential skill to have the ability to break larger work items into smaller ones in order to produce more work of higher quality. For managers of teams consisting of novice developers, we recommend training novices in this regard or pairing them with seniors from whom they can learn. Whenever possible, providing specifications as small sub-tasks, rather than larger chunks of generic features, would be beneficial in terms of improving the quality of the outcomes. These recommendations that are based on empirical evidence are also aligned with recommendations from practitioners, e.g., the elephant carpaccio analogy.

For researchers, we recommend considering the impact of programming tasks used in experimental investigations. Our results suggest that they might have an impact on the results, at least in the context of TDD experiments.

Though problem solving skills are part of the curriculum for introductory level programming courses, we recommend software engineering educators put explicit emphasis on the topic and provide practical exercises aiming to equip students with the ability to break down big problems into smaller ones.

Our findings raise further questions: Is the impact of task description granularity specific to TDD? Or is it similar for other development processes? It is possible that these findings can be applicable to other development techniques of an iterative nature. Further research on other iterative development processes are needed to test this hypothesis. In particular, comparative studies on development methods that consider factors like task description granularity would be useful to determine how specific the impact is to TDD. Therefore, we plan to extend our experimental design to incorporate iterative test-last (ITL) development in future studies.

Acknowledgments

This work was partially funded by Academy of Finland Project 278354 and Spanish Ministry of Science, Innovation and Universities research grant PGC2018-097265-B-I00.

References

[1] K. Beck, *Test-Driven Development By Example*. Boston, MA, USA: Addison-Wesley, 2003.
[2] H. Erdogmus, M. Morisio, and M. Torchiano, "On the effectiveness of the test-first approach to programming," *IEEE Trans. Softw. Eng.*, vol. 31, no. 3, pp. 226–237, Mar. 2005.
[3] L. Madeyski, "The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 169–184, 2010.
[4] D. Fucci, H. Erdogmus, and B. Turhan, "A dissection of test-driven development : Does it really matter to test-first or to test-last?" *IEEE Trans. Softw. Eng.*, vol. 43, no. 7, pp. 597–614, Jul. 2017.
[5] M. M. Müller and A. Höfer, "The effect of experience on the test-driven development process," *Empirical Softw. Eng.*, vol. 12, no. 6, pp. 593–615, Aug. 2007.
[6] A. Tosun, O. Dieste, D. Fucci, S. Vegas, B. Turhan, H. Erdogmus, A. Santos, M. Oivo, K. Toro, J. Jarvinen, and N. Juristo, "An industry experiment on the effects of test-driven development on external quality and productivity," *Empirical Softw. Eng.*, vol. 22, pp. 2763–2805, Dec. 2017.
[7] L. Madeyski, *Test-Driven Development—An Empirical Evaluation of Agile*. Berlin, Germany: Springer, 2010.
[8] S. Romano, D. Fucci, G. Scanniello, B. Turhan, and N. Juristo, "Findings from a multi-method study on test-driven development," *Inf. Softw. Technol.*, vol. 89, pp. 64–77, 2017.
[9] H. Munir, M. Moayyed, and K. Petersen, "Considering rigor and relevance when evaluating test driven development: A systematic review," *Inf. Softw. Technol.*, vol. 56, no. 4, pp. 375–394, Apr. 2014.
[10] Y. Rafique and V. B. Mi, "The effects of test-driven development on external quality and productivity: A meta-analysis," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 835–856, Jun. 2013.
[11] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus, "What do we know about test-driven development?" *IEEE Softw.*, vol. 27, no. 6, pp. 16–19, Nov. 2010.

[12] B. Turhan, L. Layman, M. Diep, H. Erdogmus, and S. Forrest, "How effective is test-driven development?" in *Making Software What Really Works, and Why We Believe It*, A. Oram and G. Wilson, Eds. Newton, MA, USA: O'Reilly Media, 2010, ch. 12, pp. 207–219.
[13] W. Bissi, A. G. Serra Seca Neto, M. C. F. P. Emer, A. G. S. S. Neto, and M. C. F. P. Emer, "The effects of test driven development on internal quality, external quality and productivity: A systematic review," *Inf. Softw. Technol.*, vol. 74, pp. 45–54, Jun. 2016.
[14] I. Karac and B. Turhan, "What do we (really) know about test-driven development?" *IEEE Softw.*, vol. 35, no. 4, pp. 81–85, Jul. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8405634/
[15] D. Sjoeberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. Rekdal, "A survey of controlled experiments in software engineering," *IEEE Trans. Softw. Eng.*, vol. 31, no. 9, pp. 733–753, Sep. 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1514443/
[16] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*, Berlin, Germany: Springer, 2000.
[17] A. J. Ko, T. D. LaToza, and M. M. Burnett, "A practical guide to controlled experiments of software engineering tools with human participants," *Empirical Softw. Eng.*, vol. 20, no. 1, pp. 110–141, 2015.
[18] H. Munir, K. Wnuk, K. Petersen, and M. Moayyed, "An experimental evaluation of test driven development vs. test-last development with industry professionals," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, 2014, pp. 1–10.
[19] B. George, L. Williams, and W. L. George B., "An initial investigation of test driven development in industry," in *Proc. ACM Symp. Appl. Comput.*, 2003, pp. 1135–1139.
[20] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mallor, K. Shwaber, and J. Sutherland, "Manifesto for Agile Software Development," 2001. [Online]. Available: https://agilemanifesto.org. Accessed: 18-Apr-2018.
[21] A. Cockburn, "Elephant carpacio," 2014. [Online]. Available: http://web.archive.org/web/20140329203040/http://alistair.cockburn.us/Elephant+carpaccio
[22] ISO/IEC 25010:2011 Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models, International Organization for Standardization ISO/IEC JTC 1/SC 7, *Tech. Rep.*, 2011. [Online]. Available: https://www.iso.org/standard/35733.html
[23] Experiment replication package, 2019. [Online]. Available: https://figshare.com/s/4882c0d634f9d821ef81, doi: 10.6084/m9.figshare.7957652.
[24] D. Fucci and B. Turhan, "A replicated experiment on the effectiveness of test-first development," in *Proc. ACM / IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2013, pp. 103–112. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6681343
[25] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. London, U.K.: Pearson Education, 2003. [Online]. Available: https://books.google.com/books?id=0HYhAQAAIAAJ&pgis=1
[26] O. Dieste, A. M. Aranda, F. Uyaguari, B. Turhan, A. Tosun, D. Fucci, M. Oivo, and N. Juristo, "Empirical evaluation of the effects of experience on code quality and programmer productivity: An exploratory study," *Empirical Softw. Eng.*, vol. 22, no. 5, pp. 2457–2542, Feb. 2017. [Online]. Available: http://link.springer.com/10.1007/s10664-016-9471-3
[27] P. Ralph, "The illusion of requirements in software development," *Requirements Eng.*, vol. 18, no. 3, pp. 293–296, 2013.
[28] C. Gonzalez, J. Dana, H. Koshinob, and M. Just, "The framing effect and risky decisions: Examining cognitive functions with fMRI," *J. Econ. Psychology*, vol. 26, no. 1, pp. 1–20, 2005.
[29] R. Mohanani, I. Salman, B. Turhan, P. Rodriguez, and P. Ralph, "Cognitive biases in software engineering: A systematic mapping study," *IEEE Trans. Softw. Eng.* (Early Access), 2018, [Online]. Available: https://ieeexplore.ieee.org/document/8506423, doi: 10.1109/TSE.2018.2877759.
[30] IEEE Computer Society. Software Engineering Standards Committee and IEEE-SA Standards Board, "IEEE recommended practice for software requirements specifications," Institute of Electrical and Electronics Engineers, standard, 1998, doi: 10.1109/IEEESTD.1998.88286.
[31] K. Schwaber, *Agile Project Management with Scrum*. Redmond, WA, USA: Microsoft Press, 2004.
[32] A. Cockburn, *Writing Effective Use Cases*, 1st Ed. Boston, MA, USA: Addison-Wesley Professional, 2000.
[33] A. Borgida, J. Mylopoulos, and R. Reiter, "On the frame problem in procedure specifications," *IEEE Trans. Softw. Eng.*, vol. 21, no. 10, pp. 785–798, Oct. 1995.
[34] C. M. Gray, S. Yilmaz, S. Daly, C. M. Seifert, R. Gonzalez, et al., "Supporting idea generation through functional decomposition: An alternative framing for design heuristics," in *Proc. Int. Conf. Eng. Des.*, 2015, pp. 309–318.
[35] T. L. Griffith and G. B. Northcraft, "Cognitive elements in the implementation of new technology: Can less information provide more benefits?" *MIS Quart.*, vol. 20, no. 1, pp. 99–110, 1996.
[36] S. S. Khan, R. L. Kumar, M. J. Khouja, and K. Zhao, "Narrow framing effects on real options: The case of it application portfolios," in *Proc. Int. Conf. Inf. Syst.*, 2010, Art. no. 186.
[37] R. Mohanani, B. Turhan, and P. Ralph, "Cognitive biases in software engineering: A systematic mapping study," *IEEE Trans. Softw. Eng. (Early Access)*, 2019, [Online]. Available: https://ieeexplore.ieee.org/document/8680661, doi: 10.1109/TSE.2019.2909033.
[38] P. Ralph and R. Mohanani, "Is requirements engineering inherently counterproductive?" in *Proc. IEEE/ACM 5th Int. Workshop Twin Peaks Requirements Archit.*, 2015, pp. 20–23. [Online]. Available: http://dblp.uni-trier.de/db/conf/icse/twinpeaks2015.html#RalphM15
[39] A. Jedlitschka and D. Pfahl, "Reporting guidelines for controlled experiments in software engineering," in *Proc. Int. Symp. Empirical Softw. Eng.*, pp. 95–104, 2005.
[40] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 456–473, Jul./Aug. 1999. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=799939
[41] D. Raghavarao and L. Padgett, *Repeated Measurements and Cross-Over Designs*. Hoboken, NJ, USA: Wiley, 2014. [Online]. Available: https://books.google.fi/books?id=t58yAwAAQBAJ
[42] D. Fucci, B. Turhan, and M. Oivo, "Impact of process conformance on the effects of test-driven development," in *Proc. 8th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Sep. 2014, pp. 10:1–10:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2652524.2652526
[43] B. Estácio, F. Zieris, L. Prechelt, and R. Prikladnicki, "On the randori training dynamics," in *Proc. 9th Int. Workshop Cooperative Human Aspects Softw. Eng.*, 2016, pp. 44–47. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2897586.2897603
[44] I. Salman, A. T. Misirli, and N. Juristo, "Are students representatives of professionals in software engineering experiments?" in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, May 2015, vol. 1, pp. 666–676. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7194615
[45] Java tutorial through katas: Mars rover, 2014. [Online]. Available: https://technologyconversations.com/2014/10/17/java-tutorial-through-katas-mars-rover/. Accessed on 30 March 2018.
[46] Mars rover exercise, 2013. [Online]. Available: https://archive.codeplex.com/?p=marsroverexercise. Accessed: 30 March 2018.
[47] D. Fucci, G. Scanniello, S. Romano, M. Shepperd, B. Sigweni, F. Uyaguari, B. Turhan, N. Juristo, and M. Oivo, "An external replication on the effects of test-driven development using a multi-site blind analysis approach," in *Proc. 10th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2016, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2961111.2962592
[48] S. Vegas, C. Apa, and N. Juristo, "Crossover designs in software engineering experiments: Benefits and perils," *IEEE Trans. Softw. Eng.*, vol. 42, no. 2, pp. 120–135, Feb. 2016.
[49] J. C. Pinheiro and D. M. Bates, *Mixed Effects Models in S and S-Plus*, 2000. [Online]. Available: http://www.amazon.com/Mixed-Effects-Models-S-S-Plus/dp/0387989579
[50] L. Madeyski and B. Kitchenham, "Effect sizes and their variance for AB/BA crossover design studies," *Empirical Softw. Eng.*, vol. 23, pp. 1982–2017, Dec. 2017. [Online]. Available: http://link.springer.com/10.1007/s10664-017-9574-5
[51] J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team, "nlme: Linear and nonlinear mixed effects models," 2018. [Online]. Available: https://cran.r-project.org/package=nlme
[52] S. Nakagawa and H. Schielzeth, "A general and simple method for obtaining R2 from generalized linear mixed-effects models," *Methods Ecology Evolution*, vol. 4, pp. 133–142, 2013.
[53] RStudio team, "RStudio: Integrated development environment for R," Boston, MA, 2015. [Online]. Available: http://www.rstudio.com/

[54] R core team, "R: A language and environment for statistical computing," Vienna, Austria. 2018. [Online]. Available: https://www.r-project.org
[55] J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team, *nlme: Linear and Nonlinear Mixed Effects Models*, 2018, R package version 3.1–137. [Online]. Available: https://CRAN.R-project.org/package=nlme
[56] H. Jacqmin-Gadda, S. Sibillot, C. Proust, J.-M. Molina, and R. Thiébaut, "Robustness of the linear mixed model to misspecified error distribution," *Comput. Statist. Data Anal.*, vol. 51, no. 10, pp. 5142–5154, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016794730600185X
[57] K. Becker, B. D. S. C. Pedroso, M. S. Pimenta, and R. P. Jacobi, "Besouro: A framework for exploring compliance rules in automatic TDD behavior assessment," *Inf. Softw. Technol.*, vol. 57, pp. 494–508, 2014.

Itir Karac received the BSc degree in mathematics and the MSc degree in computer engineering from Boğaziçi University, Turkey. She is working toward the PhD degree and is a researcher with the Empirical Software Engineering (M3S) research unit at the University of Oulu, Finland. Her research interests include empirical software engineering and software analytics. She is a member of the IEEE.

Burak Turhan, received the PhD degree from the Boğaziçi University. He is an associate professor in Cyber Security & Systems at Monash University. His research focuses on empirical software engineering, software analytics, quality assurance and testing, human factors, and (Agile) development processes. He has published more than 100 articles in international journals and conferences, received several best paper awards, and secured research funding for multiple large scale projects. He has served on the program committees of more than 30 academic conferences, on the editorial or review boards of several journals including the *IEEE Transactions on Software Engineering*, the *Empirical Software Engineering* and the *Journal of Systems and Software*, and as (co-)chair for PROMISE'13, ESEM'17, and PROFES'17. He is currently a steering committee member for ESEM, and a member of ACM, ACM SIGSOFT, IEEE and IEEE Computer Society. For more information please visit: https://turhanb.net.

Natalia Juristo has been a full professor of software engineering with the School of Computer Engineering, Technical University of Madrid, Spain, since 1997. She was awarded a FiDiPro (Finland Distinguished Professor Program) professorship at the University of Oulu, between 2013-2018. In 2009, she was awarded an honorary doctorate by Blekinge Institute of Technology in Sweden. She has been member of several Editorial Boards, including *IEEE Transactions on Software Engineering*, *Journal of Empirical Software Engineering* and *IEEE Software magazine*. For additional details and information please visit: http://www.grise.upm.es/htdocs/miembros/natalia/index.php