Test-Driven Development and Embedded Systems: An Exploratory Investigation

Michelangelo Esposito, Simone Romano, and Giuseppe Scanniello

University of Salerno, Fisciano, Italy,
{m.esposito253@studenti.unisa.it} and {siromano, gscanniello}@unisa.it

Abstract. We present the results of an exploratory investigation to obtain preliminary outcomes on the use of Test-Driven Development (TDD), an incremental approach to software development where tests are written before production code, to develop Embedded Systems (ESs). Specifically, we conducted two experiments (*i.e.,* a baseline experiment and its replication) to compare TDD with a traditional way of coding where production code is written before tests (*i.e.,*, YW, short for Your Way). The most remarkable differences between these experiments concern the implementation task and the experimental procedure. We introduced these differences in the replication to validate the results of the baseline experiment and to generalize the baseline experiment results to a different and more realistic setting. In particular, we asked the participants to implement an ES in the replicated experiment and to deploy and test it in a (real) target software/hardware environment. The developed ES was also tested before the deployment. The baseline experiment and its replication were used to gather quantitative data. We also gathered qualitative data to explain the quantitative data results to have a better picture of the phenomenon under study. Gathered data suggest that developers' productivity increases when using TDD to develop ESs, while there is not a substantial difference with respect to the external quality of the developed solutions. Developers' perspectives indicated that TDD is considered more difficult to apply as compared with YW.

Keywords: Embedded Systems, Experiment, Replication, TDD

1 Introduction

An *Embedded System* (*ES*) is a combination of hardware and software created for a particular purpose. In many cases, ESs operate as part of a larger system (*e.g.,* agricultural-sector equipment, automobiles, medical equipment, airplanes, *etc.*). The global market of ESs is expected to witness notable growth. A recent report evaluated the global market of ESs \$89.1 billion in 2021, and this market is projected to reach \$163.2 billion by 2031; with a compound annual growth rate of 6.5% [1]. This growth is related to an increase in the number of ESs-related research and advanced development projects (*e.g.,* driver-assistance systems) [1].

To date, there has yet to be shown which approach is to be preferred when developing ESs. In his book, Greening [12] asserted that embedded developers can benefit from the application of *Test-Driven Development* (*TDD*), an incremental approach to software development in which the developer repeats a short cycle, to incrementally implement functionality, made up of three phases: *Red*, *Green*, and *Refactor* [5]. In the Red phase, the developer writes a unit test for a small chunk of functionality not yet implemented and watches the test fail. In the Green phase, the developer implements that chunk of functionality as quickly as possible and watches all unit tests pass. In the Refactor phase, the developer changes the internal structure of the code while paying attention to keeping the functionality intact (*i.e.,* all unit tests have to pass). The Red-Green-Refactor cycle is thus repeated until the functionality is completely implemented. When this happens, the developer can tackle new functionality.

TDD has been conceived to develop "traditional" software and it is claimed to improve software quality as well as developers' productivity [14]. ESs have all the challenges of non-Embedded Systems (NoESs), such as poor quality, but adds challenges of their own [12]. For example, one of the most cited differences between ESs and NoESs is that the code of ESs depends on the hardware.

A huge amount of empirical investigations has been conducted to study the claimed effects of TDD when developing NoESs [14]. On the other hand, no investigation has been conducted yet to assess the benefits that TDD could bring to the development of ESs although some authors, like Greening [12], claim that the application of this approach produces benefits in the context of ESs development (*e.g.,* high-quality embedded software).

To increase our body of knowledge on the benefits deriving from the application of TDD in the context of ESs development, we investigated the following high-level Research Question (RQ):

> *Does the use of TDD affect the external quality of embedded software and embedded developers' productivity?*

To answer this RQ, we conducted an exploratory study made up of two experiments (*i.e.,* a baseline experiment and a replication) on TDD applied to the development of ESs. The participants in the study were final-year Master's students in Computer Science (CS) taking an ESs course at the University of Salerno (Italy). To assess the benefits (if any) due to the application of TDD in terms of the external (*i.e.,* functional) quality of embedded software and developers' productivity, we compared TDD with a non-TDD approach named *YW* (*Your-Way*)—*i.e.,* the approach developers would normally use to produce software when no restriction is imposed to them (except for not using TDD) [2, 3, 11]. In both experiments, whichever the used approach (TDD or YW) was to develop a given ES, the participants were asked to use *mocks* to model and confirm the interactions between the device driver and the hardware. The mocks thus intercepted the commands to and from the device by simulating specific usage scenarios [12]. In the replication, we also asked the participants to replace the mocks with the actual interactions with sensors and actuators by deploying and testing their ES in the target hardware. Such a variation in the replication was introduced to validate and generalize the results of the baseline experiment in a more-realistic development setting. In both experiments, we also gathered qualitative data to have a better picture of the phenomenon under study. Our results suggest that, overall, the external quality of the developed ESs increases when using TDD as compared to YW, while there is not a substantial difference with respect to developers' productivity. However, TDD is considered more difficult to apply to the development of ESs than YW.

Paper Structure. In Section 2, we outline related work. The design of our investigation is described in Section 3. The results are presented and discussed in Section 4 and in Section 5, respectively. We highlight the threats to the validity of our investigation in Section 6. Final remarks conclude the paper.

2 Related Work

Despite different authors (*e.g.,* [9, 12, 15, 25]) have promoted the use of TDD in the context of ESs development, nobody has ever investigated the benefits that TDD could bring to the development of ESs. On the contrary, in the context of NoESs development, the claimed benefits of TDD—including increased external quality and developers' productivity, which we investigated in our study—have been the subject of different primary studies (*e.g.,* [10, 21]), whose results have been also gathered and combined in secondary studies (*e.g.,* [6, 18, 19, 22]). For example, in their Systematic Literature Review (SLR), Turhan *et al.* [22] included 32 primary studies (2000-2009) and found a moderate effect in favor of TDD on external quality while the results about developers' productivity are inconclusive (*i.e.,* the results do not lead to a firm conclusion in favor or against TDD). Bissi *et al.* [6] conducted an SLR that included 27 primary studies (1999-2014). The authors observed, similar to Turhan *et al.* [22], an improvement of external quality due to TDD, while the results about productivity are inconclusive. Rafique and Misic [19] conducted a meta-analysis of 25 controlled experiments (2000-2011) and observed a small effect in favor of TDD on external quality, while again the results about productivity are inconclusive. Finally, Munir *et al.* [18], in their SLR, classified 41 primary studies (2000-2011) into four categories based on rigor and relevance. They found that in each category different conclusions could be drawn for both external quality and productivity.

Our investigation, although it is preliminary, has the merit to study for the first time the application of TDD in a new development context, the one of ESs, in which different authors have promoted its application (*e.g.,* [9, 12, 15, 25]). Given the preliminary nature of our study, the obtained results are not intended to be conclusive; rather, we aim to gather initial evidence on the application of TDD when developing ESs and pave the way for future research on this matter.

3 Baseline Experiment and Replication

In the following of this section, we detail the design of our exploratory study, which comprises a baseline experiment (**Exp1**) and a replicated one (**Exp2**) with the same group of participants. To design our study, we took into account the guidelines for experimentation in Software Engineering by Woholin *et al.* [24].

3.1 Research Questions

Consistent with the high-level RQ presented in Section 1, the goal of our study (formulated according to the *GQM* template [4]) is the following:

> **Analyze** the use of TDD **for the purpose of** evaluating its effects in the development of ESs **with respect to** the external quality of embedded software and developers' productivity **from the point of view of** researchers and lecturers **in the context of** an ESs course involving final-year Master's students in CS who develop ESs in Python.

Accordingly, we defined and investigated the following (low-level) RQs:

RQ1. Does the use of TDD improve the external quality of embedded software?
RQ2. Does the use of TDD increase embedded developers' productivity?

3.2 Participants

The participants in our study (*i.e.,* the baseline experiment and its replication) were final-year Master's students in CS at the University of Salerno. They were sampled by convenience among the students taking an ES course. Nine students, out of ten, accepted to participate in the study on a voluntary basis. In line with Carver *et al.*'s advice [7], we rewarded the students for their participation with two bonus points on the final mark of the course. The students were aware that: *(i)* they would receive the bonus points independently of their performance in the study; *(ii)* they could drop out of the study at any time, without being negatively judged; *(iii)* they could achieve the highest mark of the course even without participating in the study; and *(iv)* all gathered data would be confidentially treated and anonymously shared for research purposes only. The study had both research and educational goals: on one hand, we conceived the study to answer our RQs; on the other hand, the study allowed the students to gain experience with TDD applied to ESs development. As for the educational goal, TDD has attracted great attention from developers [11, 20] and it is claimed to bring benefits to the development of ESs [12].

The participants had all programming experience, and most of them rated such an experience 3 on a 5-point Likert scale (where 1 means "very inexperienced" and 5 means "very experienced"). The participants were mostly not experienced with unit testing since most of them rated 1 or 2 their unit-testing experience (again on a 5-point Likert scale). Three students had heard about TDD but nobody had a practical experience with this development approach.

3.3 Experimental Tasks

The participants had to fulfill three experimental tasks: two in Exp1 and one in Exp2. Each experimental task consisted of developing an ES for *Raspberry Pi* by coding in Python and using *unittest* and *Mock.GPIO* as testing and mocking

Table 1: Experimental task description.

Name	Description
IO	This task required developing an intelligent office system (named IO), which allows handling light and CO2 levels inside an office. To handle the light, IO relies on the information gathered from different sensors—*i.e.,* Infra-Red (IR) distance sensors, a real-time clock, and a photoresistor—and then controls a servo motor (to open/close the blinds) and a light bulb. A carbon dioxide sensor is used to monitor the CO2 levels inside the office and then control the switch of an exhaust fan.
CB	It required developing a cleaning robot (named CB) that, while moving in a room based on the commands it receives from an external system, cleans the dust on the floor along the way (by using rotating brushes). The robot also detects potential obstacles through an IR distance sensor. An intelligent battery sensor is used to check the charge left of the internal battery of the robot, and a LED is turned on to signal the need for a recharge.
SR	This task required developing a smart room system (named SR), which allowed handling light, temperature, and gas levels inside a room. To handle the light, the system relies on the information gathered from an IR distance sensor and photoresistor, and then turned on/off a light bulb. The information from a pair of temperature sensors is used to control the servo motor of a window. SR is also equipped with an air quality sensor to measure the gas levels inside the room and, if necessary, it triggers an alarm through a buzzer.

frameworks, respectively. In Exp1, the experimental tasks were: **Intelligent Office** (**IO**) and **Cleaning Robot** (**CB**). As for Exp2, the experimental task was **Smart Room** (**SR**). A brief description of the experimental tasks is reported in Table 1 (further details are available in our online material [TBW]).

For each task, the experimental material provided to the participants included: *(i)* a description of the task, which consisted of an abstract of the ES to be developed, a list of instructions, and a set of user stories describing the ES's functionality to be implemented; and *(ii)* a project template (for the *PyCharm* IDE), which contained function stubs (*i.e.,* empty functions exposing the expected API signatures), utility functions, and an example unittest test class.

Independently from the used development approach (TDD or YW) and experiment (Exp1 or Exp2), the participants when implementing an ES had to use mocks to model and confirm the interactions between the device driver and the hardware—this is similar to what Grenning suggests in his book [12]. In other words, the mocks allowed the unit tests written by the participants to simulate specific scenarios without depending on the actual sensors and actuators. We deliberately introduced a difference in Exp2 to make the implementation of the experimental task more similar to what might happen in an industrial development context [12]. In particular, we asked the participants in Exp2 (whatever the used development approach was) to apply two additional steps in the development pipeline (also named as *Embedded TDD Cycle* in the book by Grenning [12]). In the first additional step, the participants had to replace the mocks with the actual interactions with sensors and actuators by deploying their ES in the target hardware, a Raspberry Pi. Then, in the second additional step, the participants had to test the ES in the target hardware.

3.4 Independent and Dependent Variables

Regardless of the experiment, the participants were asked to fulfill each experimental task by using either TDD or YW as a development approach. Therefore, the independent variable of both Exp1 and Exp2 is **Approach**. It is a nominal variable assuming two possible values: *TDD* and *YW*. The choice of using YW as a baseline for comparison is based on past studies on TDD in the context of NoESs development (*e.g.,* [2, 3, 11]).

In both Exp1 and Exp2, the dependent variables are **QLTY** and **PROD**. The definitions of these variables are borrowed from past studies on TDD but applied to NoESs development (*e.g.,* [2, 10, 21]). The QLTY variable measures the external quality of the embedded software and, similar to the past studies mentioned above, it is defined as follows:

$$QLTY = \frac{\sum_{i=1}^{\#TUS} QLTY_i}{\#TUS} * 100 \quad (1)$$

where #TUS is the number of user stories a participant tackled, while $\mathrm{QLTY_i}$ is the external quality of the *i*-th user story tacked. A user story is tackled if at least one assert in the acceptance test suite of that user story passes. The acceptance test suites (one per user story) were pre-defined by the authors of this paper and not delivered to the participants. As for $\mathrm{QLTY_i}$, it is computed as the number of asserts passed for the i-th tackled user story out of the total number of asserts for that user story:

$$QLTY_i = \frac{\#ASSERT_i(PASS)}{\#ASSERT_i(ALL)} \quad (2)$$

QLTY assumes values between 0 and 100, where a value close to 100 indicates an ES of high external quality.

As for the PROD variable, it measures the productivity of a participant when developing an ES. Similar to past studies [2, 10, 21], PROD is defined as follows:

$$PROD = \frac{\#ASSERT(PASS)}{\#ASSERT(ALL)} * 100 \quad (3)$$

where #ASSERT(PASS) is the number of asserts passed in the acceptance test suites, while #ASSERT(ALL) is the total number of asserts in the acceptance test suites. PROD assumes values between 0 and 100, where a value close to 100 means high productivity when developing an ES.

3.5 Experimental Designs

The experimental design of Exp1 is *ABBA crossover* [23]. It is a kind of *within-participants* design where each participant receives both treatments (*i.e., A* and

Table 2: Assignment of the participants.

Approach	**Exp1**		**Exp2 (SR)**
	Period1 (IO)	**Period2 (CR)**	
TDD	P1-P4	P5-P9	P2-P3, P6-P8
YW	P5-P9	P1-P4	P1, P4-P5, P9

B). In ABBA crossover designs, there are two *sequences* (*i.e., AB* and *BA*), defined as the order with which the treatments are administered to the participants, and two *periods* (*i.e., Period1* and *Period2*), defined as the times at which each treatment is administered. The experimental groups correspond to the sequences. Also, to mitigate learning effects, each period is paired with a different experimental task. In our case, the experimental groups are *TDD-YW* and *YW-TDD*. The former first experimented with TDD to fulfill IO and then YW to fulfill CR, while the latter first applied YW and then TDD to fulfill IO and CR, respectively. The assignment of the participants to the experimental groups was done randomly: the participants P1-P4 were assigned to TDD-YW while the participants P5-P9 were assigned to YW-TDD (see Table 2).

As for Exp2, the experimental design is *one-factor-two-treatments* [24], a kind of *between-participants* design. In such a design, each participant receives only one treatment (in our case, either TDD or YW) while dealing with the same experimental task (in our case, SR). The experimental groups are paired with the treatments. The assignment of the participants to the groups was done randomly: five participants were assigned to TDD and four to YW (see Table 2). It is worth mentioning that, in both experiments, the participants had to carry out the experimental tasks alone.

3.6 Experimental Procedure

The experimental procedure of our study consisted of the following steps.

1. Recruitment. We gathered the adhesion of the students to participate in the experiments through an online questionnaire, by which we also gathered demographic information (*e.g.,* their programming experience).
2. Training. All participants attended a lesson about unit testing in Python with *unittest*. The first part of the lesson was theoretical, while the second part was practical—in particular, under lecturer's supervision, the students performed the unit testing of a NoES. Then, the participants attended a lesson about TDD, mocking (including the Mock.GPIO framework), and Raspberry Pi (including sensors and actuators). Again, the first part of the lesson was theoretical, while the second one was practical—*i.e.,* the students, with the guide of the lecturer, applied TDD to develop an ES for managing a greenhouse. Finally, as a laboratory activity, the students took part in a *warm-up task* where they all used TDD to develop an ES for managing a parking garage.

3. Exp1 Execution. We (randomly) split the participants into the TDD-YW and YW-TDD groups. Therefore, each participant applied each approach only once to fulfill IO and CR (see Section 3.5). The participants tackled the experimental tasks during two laboratory sessions, each lasting two hours, on two different days. At the beginning of each laboratory session, the participants received the experimental material (see Section 3.3), imported the project template into the PyCharm IDE, and then started implementing the user stories of the experimental task once at a time by using the requested approach. While doing so, the participants used mocks without interacting with the actual sensors/actuators (see Section 3.3). At the end of each laboratory session, the participants filled out an online post-questionnaire by which they delivered their PyCharm project and shared feedback about both experimental task and used approach (details on the post-questionnaire are not provided in the paper for space reasons).
4. Exp2 Execution. We (randomly) split the participants into the TDD and YW groups. The participants tackled the experimental task at home and each participant applied either TDD or YW to fulfill SR (see Section 3.5). The experimental material was delivered via email; once received, the participants could import the project template into the PyCharm IDE and then start implementing the user stories of the experimental task once at a time by using the requested approach. To fulfill the experimental task and similar to Exp1, the participants used mocks without interacting with the actual sensors/actuators. By the delivery date, the participants had to fill out an online post-questionnaire to deliver their PyCharm project and then book a slot for deploying and testing SR in the target hardware (at a research laboratory of the University of Salerno). The deployment of SR required each participant to replace the mocks with the actual interactions with sensors and actuators (see Section 3.3). The participants deployed and tested SR once at a time; in the end, they were (individually) interviewed by one of the authors of this paper. We conducted the interviews to gather feedback from the participants about the entire study (*i.e.,* both Exp1 and Exp1). Each interview was audio-recorded. To some extent, the variations in Exp2 reduced our control on the task execution; on the other side, these variations allowed us to mimic a real development context (*e.g.,* smart working and deployment in the target hardware).

3.7 Data Analysis

To study RQ1 and RQ2, we used boxplots to depict the distributions of the values of each dependent variable across the experimental tasks of Exp1 and Exp2 (we also plotted the mean values in these boxplots). Later, we aggregated the data from both experiments by performing a meta-analysis—to perform it, we took into account the guideline by Kitchenham *et al.* [17]. For each dependent variable and experiment, we computed the Standardized Mean Difference (SMD) as *Hedges' (adjusted) g*. To obtain joint SMDs (one for each dependent variable), we leveraged random-effects meta-analysis models. Finally, we show the results of the meta-analysis through forest plots.

As for the post-questionnaires, we used diverging stacked bar plots to summarize the answers to the closed questions (reported as statements to be rated on 5-point Likert scales) by approach.

To analyze the interview data, we first transcribed the audio recordings and then identified common themes in the interview transcripts by applying a thematic analysis. We took into account the guidelines by King [16] to identify such common themes. Briefly, we exploited the interview script (we used to guide the interviews) to develop some initial themes, and then we revised these themes as the analysis of the interview transcripts progressed until reaching a consensus on the final themes.

As regards the data analysis and presentation of its results, we limited ourselves to the essential for space reasons. Further details can be found in our online material, including analysis scripts, raw data, and experiment material [TBW].

4 Results

In this section, we first show the results regarding RQ1 and RQ2, and then those from the post-questionnaires and interviews, respectively.

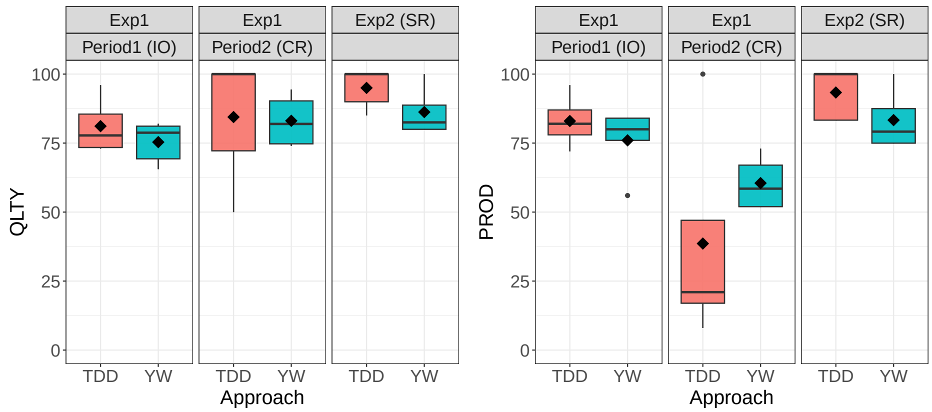
4.1 RQ1 and RQ2

In Figure 1, we show the boxplots depicting the distributions of the values of QLTY and PROD for TDD and YW across the experimental tasks (*i.e.,* IO and CR in Exp1, and SR in Exp2). The forest plots depicting, for QLTY and PROD, the SMDs across the experiments and the joint SMD are shown in Figure 2.

QLTY. The comparison with respect to QLTY between TDD and YW seems in favor of the former for any experimental task. Indeed, by looking at Figure 1.a, we can notice that the boxes for TDD are either higher than or comparable to the ones for YW. This trend is confirmed by the mean and median values (represented in the boxplots as diamonds and thick horizontal lines, respectively). We can also notice that the effect of TDD on QLTY is not uniform across the experimental tasks: the difference between TDD and YW is wider for the experimental task in Exp2, while it is more limited for the two experimental tasks in Exp1. This seems to suggest a moderating role of the experimental task. As for the SMDs, we can notice in Figure 2.a that, in the single experiments, the SMDs are both in favor of TDD. In particular, the SMD is small[1] (0.229) in Exp1 and large (0.95) in Exp2. The joint SMD is still in favor of TDD and small (0.395).

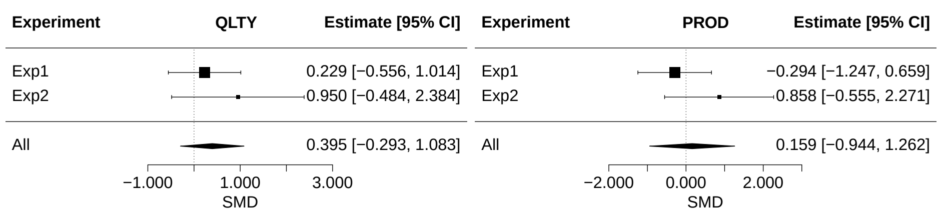
PROD. The boxplots in Figure 1.b suggest that the effect of TDD on PROD is contrasting across the experimental tasks. In particular, when considering the first experimental task in Exp1 and the one in Exp2, the boxes for TDD are either higher than the boxes for YW. On the contrary, the box for TDD is lower than the one for YW in the second experimental task of Exp1. Such a contrasting

[1] Based on Cohen's guidelines [8], the SMD can be interpreted as: *negligible*, if $|\text{SMD}| < 0.2$; *small*, if $0.2 \leq |\text{SMD}| < 0.5$; *medium*, if $0.5 \leq |\text{SMD}| < 0.8$; or *large*, otherwise.



(a) QLTY (b) PROD

Fig. 1: Boxplots with respect to QLTY and PROD.



(a) QLTY (b) PROD

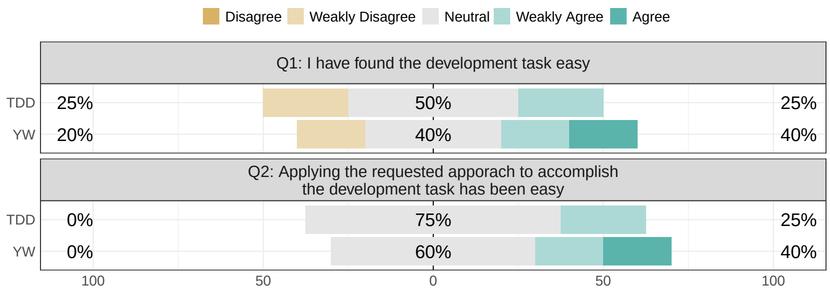
Fig. 2: Forest plots with respect to QLTY and PROD.

trend is confirmed when looking at the mean and median values, so suggesting a moderating role of the experimental task. As for the SMDs depicted in Figure 2.b, in the single experiments, the SMDs (with respect to PROD) are one in favor of YW and one in favor of TDD (Exp2). In particular, the SMD of Exp1 is in favor of YW and small (-0.294)—this is due to the second experimental task, as shown in Figure 2.b. On the contrary, the SMD of Exp2 is in favor of TDD and large (0.858). The joint SMD is still in favor of TDD but negligible (0.159).

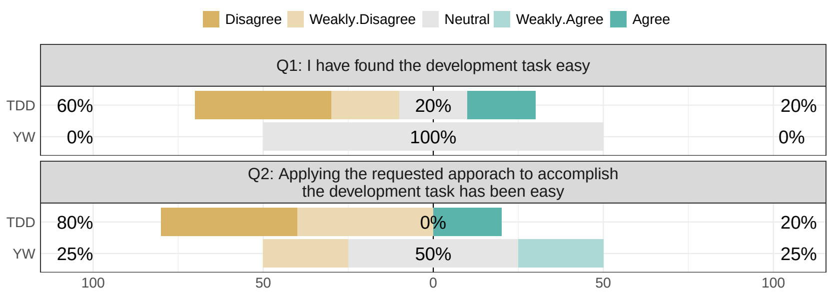
4.2 Post-questionnaires

In Figure 3, we show the diverging stacked bar plots summarizing the answers to the closed questions of the post-questionnaires the participants filled out at the end of the two experimental tasks in Exp1.

It seems that the participants who applied TDD found the experimental tasks less easy than those who applied YW. Indeed, the percentages of agreement in the first experimental task are equal to 25% for TDD and 40% for YW (see Figure 3.a). Such a difference seems stronger when considering the second



(a) First experimental task



(b) Second experimental task

Fig. 3: Diverging stacked bar plots summarizing the answers to the closed questions of the post-questionnaires in Exp1 for first and second experimental tasks.

experimental task (see Figure 3.b): the percentage of disagreement for TDD is equal to 60% while the one for YW is equal to 0%.

TDD seems to be considered less easy to be applied than YW in the first experimental task (see Figure 3.a): the percentages of agreement for TDD and YW are equal to 25% and 40% respectively. Such an outcome seems to be much stronger in the second task (see Figure 3.b) where the percentage of disagreements for TDD and YW are equal to 80% and 25%, respectively.

4.3 Interviews

In the following, we report some of the main themes resulting from the thematic analysis by highlighting them in bold. To bring credibility to our themes, we present them together with some excerpts from the interview transcripts.

1. Appreciations for Overall TDD Experience. The students appreciated the overall experience consisting of theory and practice on TDD and related topics. For example, P4 stated:

> *"I liked the overall experience made of a mix of theory and practical stuff."*

2. Appreciations for Deployment and Testing in Target Hardware. The students found it interesting and/or pleasant to see their ES running in the target hardware. For example, R2 said:

> *"Nice to see it [RS] run, and test by yourself, with hardware and sensors"*

3. Intention to Use TDD for ESs Development. The students considered TDD useful to develop ESs. Therefore, they declared they would take into account TDD to develop their ESs in the future; although some of them declared to certainly use TDD, while others on the basis of some factors (*e.g.,* the ES to be developed). An excerpt from P1's interview follows:

> *"In my opinion, TDD is better [than YW]...I would for sure use it in the future."*

4. More Practice with TDD. The students found it easier to test their production code when this is already available; therefore, they stated to need more time to be comfortable with TDD. In this respect, P6 said:

> *"In the beginning, I preferred YW because it is more natural to test the code after; however, after applying TDD in the last tasks I liked using it. However, I still need more practice with TDD."*

5 Discussion of the Results and their Implications

In this section, we outline implications for lecturers and researchers based on the main findings from our exploratory investigation.

Implications for lecturers. The data–which seem to indicate that developers' productivity improves when using TDD and that there is not a substantial difference between TDD and YW in terms of external quality–suggest properly teaching TDD. As for Exp2, we also observed by means of the descriptive statistics that TDD improves both external quality and developers' productivity. Results let us then speculate that TDD and the development pipeline [12]–which suggests writing simple simulations of hardware components (mock objects) to let tests pass and to continue writing first tests and then production code for features of a given ES–can be taught in academic ES courses instead of a more traditional approach such as YW. To some extent, this would also have practical implications from the professional perspective. That is, if a newcomer to the working market is familiar with TDD (and the studied development pipeline), the software industry could be encouraged to migrate their development from YW to TDD. In fact, software companies could be encouraged to use TDD because *(i)* there is initial evidence that it improves productivity and *(ii)* developers could be familiar with it before being hired and this reduces training and its related costs.

Participants in our exploratory investigation appreciated their overall experience in studying TDD (*e.g.,* deployment and testing of the developed solutions in a real target environment). We can postulated that this positive experience has affected the participants' intention to use TDD in the development of future ESs. This is practically relevant for lectures that should plan courses on the design and the implementation of ESs that make a mix of theory and practical experiences for the students.

Implications for researchers. The post-experiment data highlight that both the execution of the implementation tasks and the application of TDD were perceived as more difficult. To some extent, this outcome is coherent with past studies (*e.g.,* [20, 3]). A plausible explanation is that participants were more experienced with unit testing, in a test-last manner, and less accustomed to writing tests before production code. This postulation suggests two possible future research directions: *(i)* replicating our experiments with more experienced developers to ascertain that the greater the experience with unit testing in a test-last manner, the more negative their perspective with TDD is and *(ii)* conducting a study with a cohort of developers to investigate how developers perception of TDD applied to ES development changes over time.

Quantitative and qualitative results suggest that the development pipeline [12] is a valuable alternative to a traditional way of coding, where production code is written before tests. Researchers could be interested in further studying the considered development pipeline (*i.e.,* The Embedded TDD Cycle) by exploiting field studies. Our preliminary results have the merit to justify such kinds of studies.

6 Threats to Validity

To determine the threats that might affect the validity of our results, we followed Wohlin *et al.*'s guidelines [24]. Although we tried to mitigate/avoid as many threats to validity as possible, some of them are unavoidable. This is because mitigating/avoiding a kind of threat (*e.g.,* internal validity) might intensify/introduce another kind of threat (*e.g.,* external validity) [24]. Since we conducted the first study (comprising two experiments) investigating the application of TDD in the development of ESs, we preferred to reduce threats to internal validity (*i.e.,* making sure that the cause–effect relationships were correctly identified), rather than being in favor of external validity.

Construct validity. We measured each construct by a single dependent variable (*e.g.,* external quality with QLTY). As so, in case of measurement bias, this might affect the obtained results (threat of *mono-method bias*). Although we did not disclose the research goals of our study to the participants, they might have guessed them and changed their behavior based on their guess (threat of *hypotheses guessing*). To mitigate a threat of *evaluation apprehension*, we informed the participants that they would get two bonus points on the final exam mark regardless of their performance in both Exp1 and Exp2. There might be a threat of *restricted generalizability across constructs*. That is, TDD might have influenced some non-measured constructs.

Conclusion validity. We mitigated a threat of *random heterogeneity of participants* through two countermeasures: *(i)* we involved students taking the ES course allowing us to have participants with a similar background, skills, and experience; *(ii)* the participants underwent a consistent training period to make them as more homogeneous as possible within the groups. A threat of *reliability of treatment implementation* might have occurred. For example, participants might have followed TDD more strictly than others. To mitigate this threat, during the experiment, we reminded the participants to use the development approach we assigned them.

Internal validity. Participation in the experiments was voluntary and volunteers might be more motivated than professional developers. Therefore, a *selection* threat might have affected the attained results. Another threat to the internal validity is *resentful demoralization*, *i.e.,* students when assigned to a less desirable treatment might not behave as they usually would.

External validity. The participants in the experiments were Master's students, so potentially posing a threat of *interaction of selection and treatment* (*i.e.,* the results might not be generalized to professional developers). However, the use of students has the advantage that they have more homogeneous backgrounds and skills and allow obtaining initial empirical evidence [7, 13]. The implementation tasks might represent another threat to external validity: *interaction of setting and treatment*. However, we opted for ESs (*i.e.,* IO, CR, and SR) that can be considered representative of the domain of interest for our study. To deal with external validity, we also asked the participants to base the ESs implementation on a recognized development pipeline[12]. Finally, we asked the participants in Exp2 to deploy and test the ES (*i.e.,* SR) in a real environment.

7 Conclusion and Final Remarks

In this paper, we present the results of an exploratory investigation constituted of two experiments to study the external quality of ESs and developers' productivity when applying an agile software development practice like TDD. Specifically, the participants involved in the two experiments accomplished implementation tasks with Python by applying TDD or a traditional way of coding where production code is written before tests (YW). The overall results of the experiments suggest that developers' productivity increases with TDD, while there is not a substantial difference with respect to the external quality of the developed ESs. Based on the gathered data, we delineated possible implications from the perspectives of lecturers. For example, our results suggest lecturers teach TDD along with a development pipeline where hardware components are mocked before actually deploying the ES in the environment for which it has been developed. It is also worth mentioning that developers perceived TDD as more difficult than YW. This opens a number of research to better understand this phenomenon. This point is of interest to researchers.

Finally, despite we gather evidence that the TDD can be successfully applied to the development of ESs, we foster replications of our experiments, especially by involving professional developers and the software industry. Our exploratory investigation has the merit to justify such replications—it is easier to recruit professional developers when initial evidence is available. To ease the replicability of our investigation, our online material includes a replication package [TBW].

References

1. Embedded systems market by component, by application: global opportunity analysis and industry forecast, 2021-2031. Tech. rep., Allied Market Research (2022), https://www.alliedmarketresearch.com/embedded-systems-market-A08516
2. Baldassarre, M.T., Caivano, D., Fucci, D., Juristo, N., Romano, S., Scanniello, G., Turhan, B.: Studying test-driven development and its retainment over a six-month time span. J. Syst. Softw. 176, 110937 (2021)
3. Baldassarre, M.T., Caivano, D., Fucci, D., Romano, S., Scanniello, G.: Affective reactions and test-driven development: results from three experiments and a survey. J. Syst. Softw. 185, 111154 (2022)
4. Basili, V., Caldiera, G., Rombach, H.D.: Goal Question Metric (GQM) Approach, pp. 528–532. John Wiley & Sons, Ltd (2002)
5. Beck, K.: Test-driven development: by example. Addison-Wesley (2003)
6. Bissi, W., Neto, A.G.S.S., Emer, M.C.F.P.: The effects of test driven development on internal quality, external quality and productivity: a systematic review. Inf. Softw. Technol. 74, 45–54 (2016)
7. Carver, J., Jaccheri, L., Morasca, S., Shull, F.: Issues in using students in empirical studies in software engineering education. In: Proc. of International Symposium on Software Metrics. pp. 239–249. IEEE (2003)
8. Cohen, J.: A power primer. Psychol. Bull. 112, 155–9 (1992)
9. Cordemans, P., Van Landschoot, S., Boydens, J., Steegmans, E.: Test-Driven Development as a Reliable Embedded Software Engineering Practice, pp. 91–130. Springer (2014)
10. Fucci, D., Scanniello, G., Romano, S., Shepperd, M., Sigweni, B., Uyaguari, F., Turhan, B., Juristo, N., Oivo, M.: An external replication on the effects of test-driven development using a multi-site blind analysis approach. In: Proc. of International Symposium on Empirical Software Engineering and Measurement. pp. 3:1–3:10. ACM (2016)
11. Ghafari, M., Gross, T., Fucci, D., Felderer, M.: Why research on test-driven development is inconclusive? In: Proc. of International Symposium on Empirical Software Engineering and Measurement. pp. 25:1–25:10. ACM (2020)
12. Grenning, J.: Test Driven Development for Embedded C. Pragmatic Bookshelf Series, Pragmatic Bookshelf (2011)
13. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects–a comparative study of students and professionals in lead-time impact assessment. Empir. Softw. Eng. 5(3), 201–214 (2000)
14. Karac, I., Turhan, B.: What do we (really) know about test-driven development? IEEE Software 35(4), 81–85 (2018)
15. Karlesky, M.J., Bereza, W.I., Erickson, C.B.: Effective test driven development for embedded software. In: Proc. of International Conference on Electro/Information Technology. pp. 382–387. IEEE (2006)
16. King, N.: Using templates in the thematic analysis of text. In: Cassell, C., Symon, G. (eds.) Essential Guide to Qualitative Methods in Organizational Research, pp. 256–270. Sage (2004)
17. Kitchenham, B., Madeyski, L., Brereton, P.: Meta-analysis for families of experiments in software engineering: a systematic review and reproducibility and validity assessment. Empir. Softw. Eng. 25(1), 353–401 (2020)
18. Munir, H., Moayyed, M., Petersen, K.: Considering rigor and relevance when evaluating test driven development: a systematic review. Inf. Softw. Technol. 56(4), 375–394 (2014)

19. Rafique, Y., Mišić, V.B.: The effects of test-driven development on external quality and productivity: A meta-analysis. IEEE Trans. Softw. Eng 39(6), 835–856 (2013)
20. Romano, S., Zampetti, F., Baldassarre, M.T., Di Penta, M., Scanniello, G.: Do static analysis tools affect software quality when using test-driven development? In: Proc. of International Symposium on Empirical Software Engineering and Measurement. pp. 80–91. ACM (2022)
21. Tosun, A., Dieste, O., Fucci, D., Vegas, S., Turhan, B., Erdogmus, H., Santos, A., Oivo, M., Toro, K., Jarvinen, J., Juristo, N.: An industry experiment on the effects of test-driven development on external quality and productivity. Empir. Softw. Eng. 22(6), 2763–2805 (2017)
22. Turhan, B., Layman, L., Diep, M., Erdogmus, H., Shull, F.: How effective is test-driven development. In: Making Software: What Really Works, and Why We Believe It, pp. 207–217. O'Reilly Media (2010)
23. Vegas, S., Apa, C., Juristo, N.: Crossover designs in software engineering experiments: Benefits and perils. IEEE Trans. Softw. Eng. 42(2), 120–135 (2016)
24. Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer (2012)
25. Xiao, J.: Application of agile methods on embedded system development. In: Proc. of International Conference on Mechatronics, Materials, Biotechnology and Environment. pp. 667–670. Atlantis Press (2016)