An Empirical Study of Test-Driven Development vs. Test-Last Development using Eye Tracking

Joelma Choma[1], Eduardo M. Guerra[1], Tiago Silva da Silva[2], Thomas Albuquerque[2], Vanessa G. Albuquerque[3], and Luciana M. Zaina[4]

[1] National Institute for Space Research, São José dos Campos, Brazil
jh.choma@hotmail.com, guerraem@gmail.com
[2] Federal University of São Paulo, São José dos Campos, Brazil
silvadasilva@gmail.com, tealbthomas@gmail.com
[3] Group Being Educational, Guarulhos, Brazil
vanessa.ga@gmail.com
[4] Federal University of São Carlos, Sorocaba, Brazil
lzaina@ufscar.br

Abstract. Test-Driven Development (TDD) is an iterative software development technique in which unit tests are defined before production code, while Test-Last Development (TLD) is a more traditional development technique in which unit tests are written after the features are implemented. There have been a number of empirical studies investigating the effects of TDD compared to other approaches in terms of software quality and productivity. However, there are few investigations in which the TDD effects are explored from the viewpoint of the developers' experience. This paper presents an eye-tracking study carried out in order to measure visual attention during the coding and test tasks when developers are using TDD compared to TLD. Our preliminary findings pointed out a similar visual effort proportion in both techniques, but a difference regarding eye gaze behavior between them which needs to be confirmed.

Keywords: Test-Driven Development · TDD · Test-Last Development · TLD · Eye Tracking.

1 Introduction

Test-Driven Development (TDD) [3] is a technique for designing and developing software that is widely adopted by agile software development teams. TDD was proposed by Kent Beck in the late 1990s as a key practice of the Extreme Programming (XP). Popularized by XP, TDD has been considered a standalone process nowadays [26]. The single most important rule in TDD is writing test cases for what is about to code. This dynamic is referred to as "Test-First" in which the tests are used for specification purposes in addition to verification and validation. By following this practice to writing unit tests before coding, the software can be incrementally developed without a need for detailed designing it upfront, since developers are engaged to think ahead of the functionality [17].

In Test-Last Development (TLD), tests are traditionally built after the features are implemented only for verification and validation purposes [31].

Over the last decade, several empirical studies have investigated the effects of TDD compared to TLD from the perspective of developer productivity and software quality (internal and external) [14] [22] [37]. However, few empirical studies have explored the TDD effects from the viewpoint of the developers' experience [31] [32]. Software development is an intellectual activity that encompasses affective, cognitive, conative, and social aspects, going far beyond mere technical aspects [25]. Tools, techniques, methods, and development processes can be best understood to be designed or improved when studies can capture the involvement of the developers with different aspects related to the development processes, modeling methods, and other tasks [12].

In this paper, we present an eye-tracking study on developers' experience in applying TDD and TLD. This study was performed in a user experience lab involving eight developers with different experience levels and skills. An eye tracker (hardware and software) was used to monitor the developer's visual attention via eye-movements data during implementation and test tasks using TDD and TLD. We addressed two research question in this study:

- RQ1: How is visual attention distributed over implementation and test tasks when developers are using TDD compared to TLD?
- RQ2: What are the differences in eye gaze behavior between TDD and TLD presented in this study?

To answer RQ1, we choose visual effort metrics to analyze the developers' visual attention based on the two eye gaze data: (i) number of fixations and (ii) duration of fixations. Fixations refer to a focused state when the eye remains still over a while. It is a voluntary movement that can last from 200-300 milliseconds to up to several seconds. The number of fixations indicates the number of times that a user looked to a certain area of interest (AoI), and the fixation duration indicates the period that a user looked to a certain AoI. To answer RQ2, we conducted a qualitative analysis using gaze plots. A gaze plot displays a static view of the eye gaze data for each area of interest, allowing to visualize the length of the fixation and sequence of fixations (scan paths).

This paper is structured as follows: Section 2 provides related work. Section 3 describes the eye-tracking study design. Section 4 presents the study results and their analysis. Section 5 presents the discussion, conclusions and future work.

2 Related Work

2.1 Test-Driven Development

TDD process embraces a set of successive short cycles to develop the desired functionality by following three steps: (1) write a test for the next bit of functionality you want to add; (2) write the functional code until the test passes; and (3) refactor both new and old code to make it well-structured [3]. The refactoring

activity is strongly recommended in both TDD and TLD to change the production and test code and make it as simple as possible, ensuring that all tests pass [18]. Tests frameworks such as JUnit [4] were developed to enable and facilitate the implementation of unit tests. Creating unit tests is important because they help to ensure the system works correctly, mostly after code refactoring [10].

Most of the reported evidence on TDD refers to aspects of productivity (e.g., the overall time required to develop a feature), internal code quality (e.g., number of defects), and external code quality (e.g., complexity, code coverage, coupling, and cohesion between objects) [37] [36]. There are a number of controlled experiments that have been done based on objective measurements [11] [22] [14]. However, there are still few studies in which the effects of TDD are explored from the perceptions of developers [32] [8].

Gupta and Jalote [17] evaluated the impact of TDD on activities like designing, coding, and testing. Their results suggest that TDD can be more efficient regarding development efforts and the developer's productivity. Vu et al.[38] examined the TDD effects regarding software quality, and their results indicated that TDD did not outperform TLD in many quality measures. In a survey with practitioners about efficiency and quality of test cases, George and Williams [15] found that, for most of them, the TDD practice helps to create designs that are less complicated and easier to understand.

Janzen and Saiedian [20] revealed in their study that mature developers are much more likely to adopt TDD than early programmers. Scanniello et al. [32] reported that novices tend to believe more than professionals that TDD improves productivity. Munir et al. [26] by conducting a controlled experiment with professional Java developers found that the majority of participants favor TLD over TDD due to factors such as lower level of the learning curve and a minimum effort needed to maintain and understand TLD compared to TDD.

Until now, there is no consensus on results comparing the two approaches since each experience involves different contexts and potential influence factors [20]. For example, Shull et al. [36] concluded that moderate evidence exists to claim that TDD tends to improve the code's external quality, while evidence about productivity was inconclusive. For some researchers, TDD can decrease productivity because the majority of the time is devoted to the creation of tests instead of production code [24]. Some approaches involving automatic recognition systems have been proposed for conformance assessment and understanding the development behavior underlying TDD [5]. Within this context, we believe that understanding the dynamics of TDD and TLD from the developer's experience can provide important insights to researchers and practitioners, for example, improve tools or methods for training people or for supporting the development process.

2.2 Eye-tracking in Software Engineering

In software engineering (SE), the eye-tracking technology was introduced in the early nineties by Crosby and Stelovsky [9] to explore the way developers were reading an algorithm written in Pascal. Ever since then, eye-trackers have evolved

in terms of effectiveness and usability. This technology has been used mainly in human-computer interaction research [19]. Its use within research in software engineering has been restricted because of the high cost of the devices which still not easy to be gained by many researchers [27].

Sharafi et al. [34] described a set of experiments that used eye-trackers in software engineering research. More recently, Obaidellah et al. [27] also provided a mapping of the studies reporting how the experiments used eye-trackers, including information regarding experimental setup, subjects, artifacts, tasks, metrics, and type of trackers. According to these secondary studies, most engineering software researchers have used eye tracking in tasks related to model comprehension such as UML diagrams [21] [29] [35], code comprehension [7] [6] [30], and debugging [1] [13] [16].

In the agile context, Pietinen et al. [28] have investigated the interplay between pair-programming productivity and recorded developers' eye movements. In this work, they described some problems and limitations when eye tracking is used to study pair programming. In general, few eye-tracking studies consider the interaction of users with external resources, those are outside the areas of interests delimited on the computer screen.

To date, there are few studies using eye-tracking to explore more complex tasks related to development practices and processes. Moreover, most of the experiments involve simple tasks to be performed in a short time (10 to 30 minutes), and studies exploring longer processes tend to be scarce. As far as we know, no previous study has used eye-tracking to explore dynamic aspects of TDD.

3 Study Design

Following the Goal-Question-Metric (GQM) approach [2], the main goal of the study was to *analyze* the software development using TDD and TLD *for the purpose of* evaluating dynamic aspects *with respect to* the visual attention required for implementation and test from the point of view of the researcher *in the context of* user experience laboratory.

3.1 Subjects

The study participants were eight software developers from the postgraduate program at the Brazilian National Institute for Space Research (INPE). They were Master's degree students from different experience levels. As presented in Table 1, we collected the following demographic data for each participant: years of experience in programming, years of experience in Java language, level of experience in TDD. Based on this information, we divided the participants into two groups (TDD and TLD) to have two balanced groups in terms of programming experience and background.

All subjects were male and had normal vision. Only one of them wore corrective lenses. Before the study, the subjects signed an informed consent form

that provided an overview of the experimental procedures. However, they were not aware of the research questions.

Table 1. Participants' experience

Group	Subject	Programming	Java	TDD
TDD	S1	3-5 years	3-5 years	beginner
	S2	over 10 years	6-10 years	beginner
	S3	3-5 years	3-5 years	beginner
	S4	over 10 years	over 10 years	intermediate
TLD	S5	over 10 years	1-2 years	beginner
	S6	3-5 years	1-2 years	beginner
	S7	over 10 years	over 10 years	beginner
	S8	over 10 years	6-10 years	intermediate

3.2 Study Setting

The study was conducted in a user experience laboratory using the Tobii T60 eye-tracker[5] to capture different data related to eye movements and eye gaze. Through an unobtrusive data collection, the equipment provides eye gaze data which include timestamps, gaze positions, eye positions, pupil size, and validity codes. In this study, we used gaze positions and timestamps to measure visual effort. The equipment was attached and configured on a 25-inch screen (PC-1). This first screen was used by the study subjects to perform the method implementation and testing tasks, where we previously prepared the development environment. A Java code project using JUnit as a testing framework was created on the Eclipse IDE[6]. The screen was split into three areas which displayed the execution window of the unit tests, the method implementation, and the test code, placed one next to the other. On a table on the right side, we provide a 15-inch laptop (PC-2) to access the specification document and web page for searching, displayed in overlapping pages. During the study sessions, the interactions of the participants could be observed through a glass wall by the researcher who played the role of moderator.

3.3 Tasks

The task of the participants was to create in the Java language a method to transform a camelCase string into a list of strings with common words. That is, given a word as input it must be necessarily a camelCase string instance, and the output must be a list of instances of common words. Based on the pilot test,

[5] https://www.tobiipro.com
[6] https://www.eclipse.org

we defined four test cases. Table 2 presents the four input and output examples that participants should implement and use as a basis for their testing. We selected four participants to implement this task using TDD, while the remaining participants would implement it using TLD.

Table 2. Input and Output for camelCase conversion method

Test Case#	Inputs in camelCase	Outputs in String list
1	name	“name”
2	Name	“name”
3	compoundName	“name”, “compound”
4	recover10First	“recover”, “10”, “first”

3.4 Areas of Interest (AoI)

Eye movements focus a person’s visual attention to the parts of a visual stimulus when trying to understand and solve a given task [23]. In this study, the eye tracker was used to recorded eye movements and detect where the subject was looking at on the screen during the implementation and test tasks – the two visual stimuli. As shown in Figure 1, we defined two areas of interest: (AoI-1) area of implementation where the developer implements the method and (AoI-2) area of testing where the developer writes the unit tests.

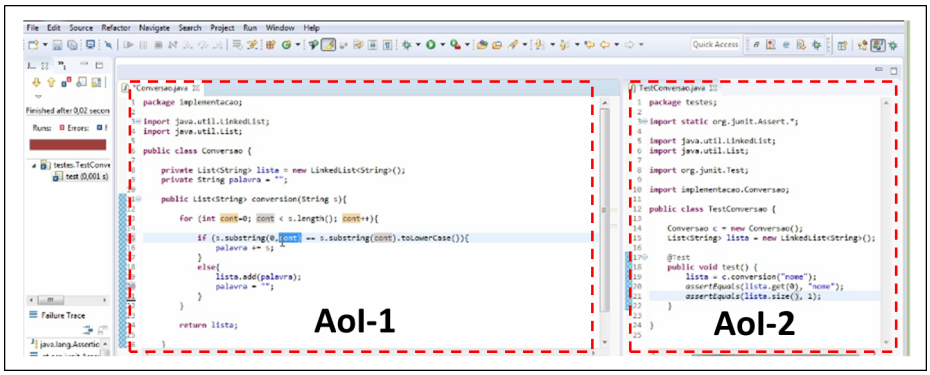

Fig. 1. Areas of interest defined on the Java project

3.5 Variables

A wide variety of eye-tracking metrics has been utilized to measure the visual effort in different types of SE tasks. Sharafi et al. [33] categorized such metrics

in four groups: (1) metrics based on fixations, (2) metrics based on saccades, (3) metrics based on scanpaths, and (4) metrics of pupil size and blink rate. However, the most common types of eye-tracking metrics are based on the number of fixations and the duration of fixation. The number of fixations indicates the number of times that a user looked to a certain area of interest (AoI), while the fixation duration indicates the period that a user looked to a certain AoI [34].

In this study, we were using six variables to cover the two areas of interest (implementation and test), which use the main two main types of eye gaze data: fixation count, the average fixation duration, and total fixation duration.

The variables are described as follows:

- Fixation Count on Implementation FC(I): The total number of eye fixations on the area of interest concerning the implementation of the camelCase conversion method. This refers to the entire method implementation.
- Fixation Count on Testing FC(T): The total number of eye fixations on the area of interest concerning unit test writing. This refers to the entire tests writing.
- Average Fixation Duration on Implementation AFD(I): The average length of time of all fixations in the area of interest concerning the implementation of the camelCase conversion method.
- Average Fixation Duration on Test AFD(T): The average length of time of all fixations in the area of interest concerning unit test writing.
- Total Fixation Duration on Implementation TFD(I): The total length of time of all fixations in the area of interest concerning the implementation of the camelCase conversion method.
- Total Fixation Duration on Test TFD(T): The total length of time of all fixations in the area of interest concerning unit test writing.

The first two measures are based on eye fixations, where a higher fixation count indicates more effort needed by subjects to solve the task. The last four measures are based on eye fixation duration, where the more time spent implementing the method or writing the tests indicates more effort needed by subjects to solve the task. The unit of measure is seconds.

3.6 Procedure

At the beginning of each section, the moderator performed the eye-tracker calibration for each participant. During the calibration, five points are displayed on the screen and mapped their locations with the coordinates of the participants' eye movements. With some participants, the calibration was repeated one more time with of purpose to achieve the highest possible accuracy.

After calibration of the equipment, the next screen displayed instructions on the task. The moderator guided the participants regarding the task to be performed in PC-1 and showed the requirements document and the web page for searching located in the PC-2. The moderator clarified to the participants that only general doubts about java syntax were allowed to be searched from

the internet. Also, it was established that, participants could only use basic Java language API classes for method development. Therefore, the use of external components was not allowed.

Finally, the participants were told that they would have approximately one hour to complete their tasks. After development activity, we had them fill out a post-test questionnaire, with the objective of gathering their perceptions about the technique used in the development (TDD or TLD) and the difficulties they encountered during the execution of the task.

4 Results

The results of the study in terms of developers' performance showed that only three participants were able to complete the task successfully developing all test cases, with one performed the tasks in 22 minutes (S3), while the other two took just over an hour (S4 and S8). In the TDD group, the participant S1 did not perform the last two test cases, and the participant S2, who had a great potential to complete the entire task, forgot to implement the test case 2 or did not do it for some reason still unknown. In the TLD group, the participant S6 was unable to conclude the last test case. Unfortunately, two participants who had used TLD (S5 and S7) were unable to complete any test cases.

Table 3 shows the time spent in minutes and the proportional time (%) spent on each test cases (TC) performed by each participant. Proportional time for each test case (see Table 2) was computed as a ratio of time spent on a test case to the overall time spent on the task. By analyzing the proportion of dedicated time, we intended to identify which test cases were easier or more difficult to solve. However, the results indicated that the participants of the two groups had different degrees of difficulty.

Table 3. Time spent in minutes and the proportional time (%)

Group	Subject	Time Spent	TC1	TC2	TC3	TC4
TDD	S1	75	0.61	0.39	–	–
	S2	57	0.30	–	0.59	0.11
	S3	22	0.35	0.11	0.50	0.04
	S4	64	0.08	0.02	0.41	0.49
TLD	S5	75	1.00	–	–	–
	S6	76	0.21	0.09	0.22	0.48
	S7	74	1.00	–	–	–
	S8	67	0.55	0.04	0.03	0.38

4.1 Visual Attention Analysis (RQ1)

To verify how developers' visual attention was distributed over implementation and test tasks (RQ1), we analyzed the visual effort metrics related to the number of fixations (FC) and the duration of fixation (AFD and TFD). Table 4 presents the results, where the values in brackets (next to the fixation count values) refer to the proportion of fixations in each area of interest concerning the total fixations individually captured. The proportional fixation count for each area was computed as a ratio of fixation count on an area to the overall fixation time on two areas. On average, in terms of fixation count, participants in both groups had approximately 76.2% visual effort on the area of implementation and 23.8% visual effort on the test area.

As for AFD(I), we found lower fixation gaze time by participant S3 the one who solved the task faster using TDD. Participant S8 using TLD had the longest fixation time in the implementation area. Concerning AFD(T), two participants using TDD (S1 and S3) and two participants using TLD (S5 and S6) had a longer fixation time in the test area than implementation area. However, concerning total fixations duration (TFD), all participants in the two groups had a longer total fixation gaze duration in the area of implementation. The visual effort of the participant S6 using TLD was almost twice as high as the other participants, both in the implementation and in the test.

When analyzing the TFD(I) results of the two subjects who successfully completed the task almost within the same time (S4 and S8), we found both developers had a similar total fixation gaze duration in the area of implementation. However, the total fixation duration in the test area of the participant who used TDD (S4) was significantly lower than the participant who used TLD (S8).

Table 4. Number of fixations and the duration of fixation metrics

Group	Subject	FC(I)	FC(T)	AFD(I)	AFD(T)	TFD(I)	TFD(T)
TDD	S1	2414 (59.6%)	1636 (40.4%)	0.23	0.33	584.16	359.19
	S2	2667 (71.6%)	1058 (28.4%)	0.13	0.04	668.66	169.71
	S3	1177 (64.1%)	659 (35.9%)	0.05	0.07	251.2	122.45
	S4	3632 (96.4%)	137 (3.6%)	0.22	0.17	899.67	29.03
TLD	S5	1143 (86.9%)	173 (13.1%)	0.1	0.15	135.8	19.9
	S6	5492 (77.3%)	1613 (22.7%)	0.09	0.17	1680.5	488.57
	S7	3324 (77.0%)	993 (23.0%)	0.17	0.17	779.02	221.66
	S8	4329 (81.4%)	988 (18.6%)	0.25	0.14	982.41	241.07

FC-Fixation Count|AFD-Average Fixation Duration|TFD-Total Fixation Duration
I - Implementation of the method | T - Test code

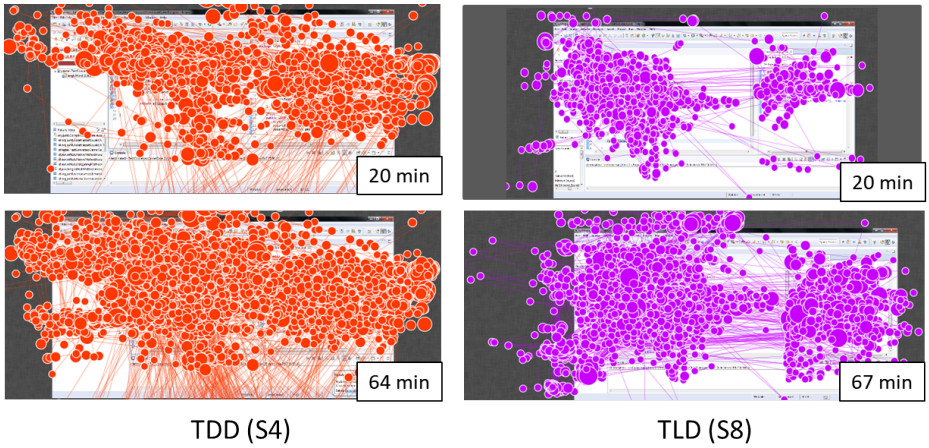

Fig. 2. Static view of the eye gaze for implementation and test from TDD and TLD.

4.2 Qualitative analysis (RQ2)

To verify differences in eye gaze behavior by comparing TDD and TLD (RQ2), we only analyzed the gaze plots of the two subjects who completed the task successfully in similar timing (S3 and S4), respectively using TDD and TLD. Figure 2 shows two gaze plots for each developer. The first two gaze plots display the static views after 20 minutes of developing, and the last two plots display the static views in the final time of completion of tasks. As shown in the gaze plots, the circles represent the fixation dots, which the radius is proportional with the fixation duration, while the lines represent the eye movements (saccades) which connect fixation dots.

When comparing TDD and TLD plots, we found two different behavior. The interaction between both implementation and test tasks seems to be more intense in TDD than TLD. That is, the fixation dots cover the two AoIs more uniformly in the gaze plots of TDD. While, in the TLD, the gaze plots showed a larger space not filled between the two AoIs, which to allow us to notice that implementation and tests were worked in separate times.

5 Discussion and Conclusion

The study was designed to be completed in less than 60 minutes to avoid the fatigue effect [30]. With more time, maybe more subjects could have been able to complete their tasks. We sought to balance the distribution of the subjects based on their experience levels. However, when analyzing the performance of participants, we noticed that each subject had its own pace of development and different skills for problem-solving. We observed that different experience levels, preferences, and reasoning processes could impact on rhythms and patterns of

development, as pointed out by Wang and Erdogmus [39]. Furthermore, programmers tend to define the best way to work in a given context.

We analyzed the subjects' behaviors individually by using descriptive statistics rather than inferential one because the sample was small. We recognized that the low number of participants did not favor ensuring well-balanced groups, since in the TLD group, for instance, only one developer was able to solve the task successfully. To mitigate this threat, we could increase the statistical power of the study. However, finding experienced TDD practitioners who are available to participate in this kind of research is a challenge [5].

As for the distribution of visual attention (RQ1), the fixation count results suggest that, on average, the implementation and testing effort have similar proportion in both development techniques (TDD and TLD). At first glance, the average duration of the fixations seems to be directly related to the subjects' skills. The subject with the best performance (S3) had a low fixation gaze time in the two areas of interest. However, something that caught our attention was the fact that this particular participant had declared experience of fewer than 5 years (both programming and Java) and to be a beginner in TDD. In the post-test questionnaire, this participant stated had no problem with implementing TDD. He further stated that TDD helped him in the implementation of the proposed tasks since he was induced to implement it in parts, completing the tasks without drawbacks.

Another participant who completed the task using TDD (S4) is a more experienced programmer and with an intermediate knowledge in TDD. However, his fixation gaze time was higher than the fixation gaze time of the S3. In the post-test questionnaire, he also stated no problem during the tasks. Within the same group, by comparing the two subjects (S3 and S4), we noticed that the participant with more experience in TDD had a low number of fixations and also a smaller total fixation duration in the area of the test (AoI2). Analyzing the individual behavior, we could see that S4 had a greater effort in writing the tests at the beginning of the development. However, this effort was softened over time, and then his focus turned to the implementation of the method. Nevertheless, further investigations will be needed to verify if this fact is related to the experience background with TDD.

About the difference in eye gaze behavior between TDD and TLD (RQ2), we found differences by observing static views of two of the participants. However, such behavior needs to be verified in future work by considering a larger sampling. If such behavior is confirmed as a pattern between the two techniques, we can try to investigate the impact of each technique from the point of view of cognitive effort, for example.

Overall, existing studies when comparing TDD and TLD focus their analyses on the final results of the code and whether the tests were written first or not. In contrast to this, we are interested in understanding the dynamics of the development process underlying the two techniques. Thus, we focused on the dynamic part of the development process and the developers' activity during the tasks. The main contribution of this study is to characterize the TDD and TLD

techniques from the developers' experience. This paper presents our preliminary results in this direction.

In future work, more studies are needed to confirm the findings on development patterns meet when we analyzed the eye gaze data using gaze plots. Further, we intend to analyze the developers' visual attention considering the external resources available in the study environment such as the requirements specification document and searching sources.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Also, we would like to thank the support granted by Brazilian funding agency FAPESP (grant 2014/16236-6 and 2014/25779-3, São Paulo Research Foundation).

References

1. Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., Parnin, C.: Do developers read compiler error messages? In: Proceedings of the 39th International Conference on Software Engineering. pp. 575–585. IEEE Press (2017)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. Encyclopedia of Software Engineering pp. 528–532 (1994)
3. Beck, K.: Test-driven development: by example. Addison-Wesley Professional (2002)
4. Beck, K.: JUnit Pocket Guide: Quick Look-up and Advice. " O'Reilly Media, Inc." (2004)
5. Becker, K., Pedroso, B.d.S.C., Pimenta, M.S., Jacobi, R.P.: Besouro: A framework for exploring compliance rules in automatic tdd behavior assessment. Information and Software Technology **57**, 494–508 (2015)
6. Bednarik, R., Tukiainen, M.: Analysing and interpreting quantitative eye-tracking data in studies of programming: Phases of debugging with multiple representations. In: Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group (PPIG'07), Joensuu, Finland. pp. 158–172. Citeseer (2007)
7. Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J.H., Schulte, C., Sharif, B., Tamm, S.: Eye movements in code reading: Relaxing the linear order. In: Proceedings of the 23rd International Conference on Program Comprehension. pp. 255–265. IEEE (2015)
8. Choma, J., Guerra, E.M., da Silva, T.S.: Developers' initial perceptions on tdd practice: A thematic analysis with distinct domains and languages. In: International Conference on Agile Software Development. pp. 68–85. Springer (2018)
9. Crosby, M.E., Stelovsky, J.: How do we read algorithms? a case study. Computer **23**(1), 25–35 (1990)
10. Deng, C., Wilson, P., Maurer, F.: Fitclipse: A fit-based eclipse plug-in for executable acceptance test driven development. In: International Conference on Extreme Programming and Agile Processes in Software Engineering. pp. 93–100. Springer (2007)
11. Desai, C., Janzen, D., Savage, K.: A survey of evidence for test-driven development in academia. ACM SIGCSE Bulletin **40**(2), 97–101 (2008)

12. Fagerholm, F., Münch, J.: Developer experience: Concept and definition. In: Proceedings of the International Conference on Software and System Process. pp. 73–77. IEEE Press (2012)
13. Fritz, T., Begel, A., Müller, S.C., Yigit-Elliott, S., Züger, M.: Using psycho-physiological measures to assess task difficulty in software development. In: Proceedings of the 36th International Conference on Software Engineering. pp. 402–413. ACM (2014)
14. 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: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. p. 3. ACM (2016)
15. George, B., Williams, L.: A structured experiment of test-driven development. Information and Software Technology **46**(5), 337–342 (2004)
16. Goswami, A., Walia, G., McCourt, M., Padmanabhan, G.: Using eye tracking to investigate reading patterns and learning styles of software requirement inspectors to enhance inspection team outcome. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. p. 34. ACM (2016)
17. Gupta, A., Jalote, P.: An experimental evaluation of the effectiveness and efficiency of the test driven development. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement. pp. 285–294. IEEE Computer Society (2007)
18. Ivo, A.A., Guerra, E.M.: Retest: framework for applying tdd in the development of non-deterministic algorithms. In: Brazilian Workshop on Agile Methods. pp. 72–84. Springer (2016)
19. Jacob, R.J., Karn, K.S.: Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. In: The mind's eye, pp. 573–605. Elsevier (2003)
20. Janzen, D.S., Saiedian, H.: A leveled examination of test-driven development acceptance. In: Proceedings of the 29th International Conference on Software Engineering (ICSE'07). pp. 719–722. IEEE (2007)
21. Jeanmart, S., Gueheneuc, Y.G., Sahraoui, H., Habra, N.: Impact of the visitor pattern on program comprehension and maintenance. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement. pp. 69–78. IEEE Computer Society (2009)
22. Jeffries, R., Melnik, G.: Guest editors' introduction: Tdd–the art of fearless programming. IEEE Software **24**(3), 24–30 (2007)
23. Kanwisher, N., Wojciulik, E.: Visual attention: insights from brain imaging. Nature reviews neuroscience **1**(2), 91 (2000)
24. Khanam, Z., Ahsan, M.N.: Evaluating the effectiveness of test driven development: advantages and pitfalls. International Journal of Applied Engineering Research **12**(18), 7705–7716 (2017)
25. Kuusinen, K., Petrie, H., Fagerholm, F., Mikkonen, T.: Flow, intrinsic motivation, and developer experience in software engineering. In: International Conference on Agile Software Development. pp. 104–117. Springer (2016)
26. Munir, H., Wnuk, K., Petersen, K., Moayyed, M.: An experimental evaluation of test driven development vs. test-last development with industry professionals. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. p. 50. ACM (2014)

27. Obaidellah, U., Al Haek, M., Cheng, P.C.H.: A survey on the usage of eye-tracking in computer programming. ACM Computing Surveys (CSUR) **51**(1), 5 (2018)
28. Pietinen, S., Bednarik, R., Tukiainen, M.: Shared visual attention in collaborative programming: a descriptive analysis. In: Proceedings of the Workshop on Cooperative and Human Aspects of Software Engineering. pp. 21–24. ACM (2010)
29. Porras, G.C., Guéhéneuc, Y.G.: An empirical study on the efficiency of different design pattern representations in uml class diagrams. Empirical Software Engineering **15**(5), 493–522 (2010)
30. Rodeghero, P., McMillan, C., McBurney, P.W., Bosch, N., D'Mello, S.: Improving automated source code summarization via an eye-tracking study of programmers. In: Proceedings of the 36th International Conference on Software Engineering. pp. 390–401. ACM (2014)
31. Romano, S., Fucci, D., Scanniello, G., Turhan, B., Juristo, N.: Results from an ethnographically-informed study in the context of test driven development. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. p. 10. ACM (2016)
32. Scanniello, G., Romano, S., Fucci, D., Turhan, B., Juristo, N.: Students' and professionals' perceptions of test-driven development: a focus group study. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. pp. 1422–1427. ACM (2016)
33. Sharafi, Z., Shaffer, T., Sharif, B., Guéhéneuc, Y.G.: Eye-tracking metrics in software engineering. In: Software Engineering Conference (APSEC), 2015 Asia-Pacific. pp. 96–103. IEEE (2015)
34. Sharafi, Z., Soh, Z., Guéhéneuc, Y.G.: A systematic literature review on the usage of eye-tracking in software engineering. Information and Software Technology **67**, 79–107 (2015)
35. Sharif, B., Maletic, J.I.: An eye tracking study on the effects of layout in understanding the role of design patterns. In: Software Maintenance (ICSM), 2010 IEEE International Conference on. pp. 1–10. IEEE (2010)
36. Shull, F., Melnik, G., Turhan, B., Layman, L., Diep, M., Erdogmus, H.: What do we know about test-driven development? IEEE Software **27**(6), 16–19 (2010)
37. Turhan, B., Layman, L., Diep, M., Erdogmus, H., Shull, F.: How effective is test-driven development. Making Software: What Really Works, and Why We Believe It pp. 207–217 (2010)
38. Vu, J.H., Frojd, N., Shenkel-Therolf, C., Janzen, D.S.: Evaluating test-driven development in an industry-sponsored capstone project. In: Proceedings of the Sixth International Conference on Information Technology: New Generations. pp. 229–234. IEEE (2009)
39. Wang, Y., Erdogmus, H.: The role of process measurement in test-driven development. In: Conference on Extreme Programming and Agile Methods. pp. 32–42. Springer (2004)