Test-Driven Development Benefits Beyond Design Quality: Flow State and Developer Experience

Pedro Calais
Stone Co.
Rio de Janeiro, RJ, Brazil
pedro.guerra@stone.com.br

Lissa Franzini
Stone Co.
Rio de Janeiro, RJ, Brazil
lissa.franzini@stone.com.br

***Abstract*—Test-driven development (TDD) is a coding technique that combines design and testing in an iterative and incremental fashion. It prescribes that tests written *before* the production code help the developer to find good interfaces and to evolve the design safely and incrementally. Improvements on the design of code produced by the test-driven development approach have been extensively evaluated in the literature; in this research, we focus on seeking explanations on the benefits of TDD in another dimension which we believe has been undervalued – *developer experience*. We identified that there is a natural connection between the TDD approach and *flow state*, a well-known mental state characterized by total immersion, focus, and involvement in a task that promotes increased enjoyment and productivity. We present evidence that the continuous stream of mini-scope, short-lived, red-green-refactor cycles of TDD frame the development task as a structure that creates the pre-conditions reported by neuroscience research to produce flow state, namely (1) clear goals, (2) unambiguous feedback, (3) challenge-skill balance and (4) sense of control. Our work contributes to increase the understanding on the reasons why adopting practices such as TDD can benefit the software development process as a whole and can support its adoption in software development projects.**

***Index Terms*—TDD, developer experience, flow state**

I. Introduction

Test-driven development (TDD) is a popular software development technique that mixes testing, coding and design in an incremental and iterative fashion [1]. In the TDD approach, unit tests are written *before* the actual code that will make the test pass (see Figure I). Although initially a counter-intuitive approach, TDD practitioners claim that starting by writing the tests help to better focus on the requirements and to produce simpler and better code, with the bonus side-effect that code produced by TDD is also always unit-tested [2]. The intuition is that TDD tends to produce more decoupled code because tests written before the code focus on the desired behavior (rather than on the implementation details) and hence it drives testable, simpler code than other approaches, such as test-last-development [3]. There is plenty of anecdotal and informal evidence that TDD produces code of superior quality, while more rigorous empirical and experimental evaluations diverge in finding some impact [4] or no difference at all [5].

Whilst case studies and experiments on TDD have largely focused on code quality and productivity metrics [6]–[9], in this research we focus on another dimension of the software development process, namely the *developer experience*, i.e., the emotions, thoughts and feelings of developers as they perform their activities [10]–[13], an aspect of software engineering that strongly impacts job satisfaction, team performance and the business as a whole [12]–[15].

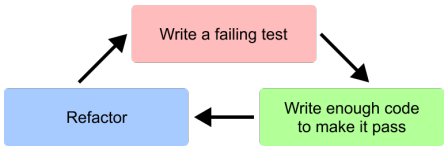

Fig. 1. The test-driven development technique. In TDD, the programmer starts by writing a failing test (the red step). Next, he or she writes code that is just enough to make the test pass (the green step). Finally, as the programmer learns about the regularities and properties of the problem space, refactors can be implemented that keeps the code clean (the blue step). Notice the refactor step is safe since it will always be backed up by tests.

More specifically, we seek to increase the understanding of the software development community on the benefits of TDD in terms of its impact in the developer's mental state; to do that, we connect the TDD recipe with a well-established cognitive theory named *flow state* [16]. Flow state is a mental state one can experience when intensively involved in an activity with high degrees of concentration and focused attention accompanied by a sense of pleasure [17]. It is formally characterized as a state of full task engagement [18] and a "holistic sensation that people feel when they act with total involvement", as first presented by Csikszentmihalyi in [19].

The benefits of flow for enhanced performance and well-being have been widely documented [20], [21] and a wide body of knowledge has found evidence that well-known brain systems related to attention processes [22], emotional arousal [23], self-awareness [24] and reward [21] play a role in the flow experience. The current state of knowledge about flow confirms its association with boosts in performance and positive mood states, thus being highly relevant for improving work experience in various domains, especially due to its characteristic of promoting a mental state where increased performance and creativity occur in a context of high engagement and well-being [16], [25].

Flow state has been reported in a variety of physical and mental activities, including sports [26], games [27]–[29] and arts [30]. More connected to our work, flow also commonly arises during work [31] and arithmetic tasks [23]. Some

works connect flow with software engineering [32]–[34], while developers reporting being in the flow state while writing code is reported in some sources [35]. To the best of our knowledge, we are the first to connect the practices of TDD with the detailed mechanics that favors one to enter the flow state.

Our main hypothesis is that test-driven development sees finding a good solution for a development task as a *learning and discovery process* [36], [37], and the tests act as a guide in that process. As such, TDD creates a structure in the development task that helps to induce the flow state in the developer, i.e, the mental state of high productivity and sense of well-being. We present three evidences that show that, in our view, connecting test-driven development and software development strategies in general with neurocognitive theories that describe how our brain learn and operate can be a promising and relevant research topic:

1) First, we present a theoretical relationship between the structured coding process prescribed by TDD (see Figure I) and the pre-conditions (or triggers) shown by extensive research to induce in our brains feelings of extra focus, productivity and enjoyment (*flow state*).
2) Second, we ran a survey with an adapted and shortened version of the flow-state scale questionnaire, a standard practice in psychology to measure flow state in a 1-5 scale [38]. Our results show that TDD practitioners seem to experience flow state, and more experienced developers experience it at a higher intensity;
3) Finally, we show some self-report data collected from popular online tech forums and blog posts that show that developers informally and indirectly refer to experiencing flow state when adopting TDD.

Next, we will provide details on each of those evidences.

II. Evidence #1: TDD structure and Flow Triggers

The neuroscience literature describes some key triggers, or pre-conditions, that, when present, induce flow state to some extent [39]–[42]: 1) clear goals, 2) unambiguous feedback and 3) challenge-skill balance. Here, we also include sense of control as a fourth trigger, in accordance to more recent findings [20] indicating that, although being a feeling experimented as a consequence of flow, it can also be as powerful as challenge-skill balance as a cause of the flow state. For each flow pre-condition, we summarize the reasons found by neuroscience literature that make it a trigger of flow, and then explain how the TDD red-green-refactor process, as an intellectual task, enables each one of those flow pre-conditions.

A. Flow trigger #1: Clear Goals

> [TDD] is a predictable way of develop. **You know when you are finished**, without having to worry about a long bug trail [1].

.

Clear goals might generate flow by showing you where and when to put your attention [43] and affect the need for active cognitive control and reduces felt effort [23], [29]. The benefits our brain enjoy when the goals are clear is that distraction and cognitive load are reduced, as unnecessary information is filtered out. Our concentration tightens, motivation increases, action and awareness merge, and we are pulled into flow – where we think and perform best [25].

In the red-green-refactor TDD cycle (see Figure I), the red step is the one that concretely guides the developer into establishing and following a clear goal: by prescribing that the test should be written *before* production code, it forces the programmer to define and formulate clearly what he or she wants to accomplish and think on the problem formulation and why it is relevant, rather than rushing to the solution. Without a goal, it is simply not possible to start the TDD process.

Actually, note that the whole red-green-refactor cycle guides the programmer in the sense that he or she naturally knows what to do next: the programmer either has to write a next test that fails (which tends to be the failing test that captures the next natural dimension of the problem – the red step), or make the current failing test pass (the green step). Without tests (written before the code) that encode the goal to be reached, the absence of a clear goal makes it easier to disperse and lose focus. On the other hand, accomplishing a goal (making a test pass) can release dopamine, a neurotransmitter involved in the brain's reward system [44].

B. Flow trigger #2: Unambiguous feedback loops

> We must design organically, with running code **providing feedback** between decisions [1].

When performing a task, having immediate feedback helps the nervous system on performance monitoring, continuously providing information about whether the challenge-skill balance is optimal or not [21], allowing for adjustments to be made to ensure that one's performance matches one's goals. The presence of a feedback loop in an activity, according to Csikszentmihalyi [16], promotes happiness even though those activities may not seem enjoyable at first; but once the interaction starts to provide positive feedback to the person's skills, it usually begins to be intrinsically rewarding, encouraging the person to keep working.

The introduction of feedback loops is actually a foundational principle of extreme programming and agile principles. In TDD, each unit test provides feedback to the programmer; the green tests indicate the part of the logic which works as expected, while failing tests provides guidance on where one needs to focus, guiding the programmer into navigating the flow channel. State-of-the-art unit test practices advocate that tests should be fast [45], which is a strategy to shorten the feedback loop. Let us point out the specific steps of the TDD process where feedback is provided to the programmer.

- The red step – writing a test that fails – provides a feedback on the requirement **testability**: if the programmer has a hard time writing the test, it means that the requirement needs further clarification. Furthermore, calling the code from the test provides feedback on the

cleaniness of the interface since the test is the code API's first user.

- Furthermore, TDD's red step provides feedback on the **test falsifiability**; running a new failing test indicates to the programmer that he or she is actually evolving the design to capture a new use case and that the test is valid, and it does not overlap with an existing test.
- The green step – running the tests and checking that they are green – provides feedback on **code correctness**, including after refactorings.

C. Flow trigger #3: Challenge-skill balance

> TDD gives you a chance to learn all of the lessons that the code has to teach you [1].

Every activity one engage in, be it leisure or work related, falls somewhere in a spectrum of how challenging it is to the person and the level of skills it demands. A lack of balance between skill level and challenge limits enjoyment and productivity. If the challenge is too demanding, we get frustrated. If it is too easy, we get bored. One of the key findings of research in flow experiences is that flow is usually experienced when one navigates in the *flow channel* [19], shown in Figure II. It shows that when the challenge and the skill required to tackle it are proportional, we feel engaged and motivated by the challenge, but not overwhelmed.

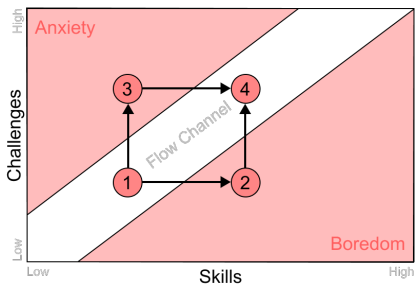

Fig. 2. Flow is dependent on the balance of the difficulty of a task and skill. When the challenge and skill are proportionate (zones 1 and 4) one can navigate through the flow channel. When the challenge is too small compared to skill level (zone 2), the person will likely get bored. When the challenge is too high compared to the skill level, the person will likely get anxious (zone 3). The most desirable state is entered when skill and difficulty are proportionate.

Let us examine how TDD tends to keep the mental effort of the programmer inside the flow channel and hence promotes challenge-skill balance. First, note that a complex problem whose solution is unknown a priori by the programmer, especially one that not only works, but concretizes an acceptable design, maybe be hard to find even if you are an experienced programmer. So, development approaches such as TLD (test-last development) and no-tests at all tend to, in terms of mental state, stay in zone 2 (anxiety) in Figure II. A complex problem imposes a high cognitive load on the programmer, something that relates to stress [46] and thus move the programmer away from flow state because it demands attention to too many things, moving him or her away from a state of mental clarity. When tackling a complex problem, we do not usually own complete knowledge about it, and the programmer is compelled to think of the problem as a whole, i.e, the psychological setting is one of entropy and disorder, likely to produce anxiety [47].

In the TDD red-green-refactor loop, the programmer will incrementally think of the problem one test case at a time and the challenge tends to increase since the code has to accommodate the various test cases. However, the knowledge about the problem's regularities also increases, and thus the skills needed to solve the problem. This process of incrementally and iteratively finding a solution is strongly connected to findings that the feeling of flow is not a constant: if a person's skill level improves, the challenge of the task needs to increase accordingly to keep the individual in flow [21]. Note that the programmer has passing tests to rely on at all times, what guides him or her into staying inside the flow channel since, if the programmer feels the new test is forcing a step that is not clear enough, a simpler test can be created to keep him or herself inside the flow channel.

In summary, the TDD step-by-step process optimize the programmer's work for learning [37], what connects nicely to successful reports of teachers intentionally trying to design learning activities that seek challenge-skills balance and recognizing that students' skills need to match the goal since some problems are too complex to be tackled at once [48].

D. Flow trigger #4: Sense of control

> [TDD] can help you become a more effective and disciplined developer – **fearless** and joyful! [5]

Sense of control provides the perception that the person engaging in the task is able to respond to the challenge in hand, and hence it can be a powerful source of flow and a key factor for peak performance [21] and motivation [20]. In TDD, tests work as a safety net, not only during development, but after it. The programmer can quickly learn whether the system is working or not by running the test suite [49]. On the other hand, in TLD, the programmer cannot easily learn whether the code he or she just wrote works, what can move him or her away from the sense of control and hence flow experience. Furthermore, since the programmer can always resort back to the last commit, one feels a sense of control without actively trying to be in control as he or she explores new designs in the refactor step.

III. Evidence #2: The Flow-State Questionnaire

Given the challenges in studying and identifying patterns in human cognition, in addition to a theoretical discussion, we also seek empirical and experimental evidences that test-driven development is likely to promote a sense of well-being and boost developer's productivity. One method used to quantitatively measure flow is to apply questionnaires such as the flow-state scale questionnaire [38], which is comprised of a set of questions and answers given in a 5-point scale. We adapted each question to the TDD context; as an example, the question "I felt I could control what I was doing" has been adapted to "Guiding the design by tests helps me feel

Flow-State-Scale Question	Avg. Response (experts)	Avg. Response (intermediate)
I feel that when I write tests in a TDD-like structure, i.e, interleaving tests with the code, it helps me knowing clearly what I need to do in order to solve the coding problem.	4.7	3.9
I feel that writing the tests before the actual code helps me feel more confident while coding and makes me feel more competent to deal with complex coding problems.	4.6	4.0
Compared to when I do not write tests or write tests after developing the code, I feel the coding experience with TDD is more rewarding.	4.5	3.9
Guiding the design by tests helps me feel like I could control what I am doing.	4.5	3.7
Compared to when I do not write tests or write tests after developing the code, when following the TDD recipe I feel I have a stronger sense of what I wanted to do and knew how well I was performing in the coding activity.	4.5	3.6
Compared to when I do not write tests or write tests after developing the code, I feel the coding experience with TDD is more rewarding.	4.5	3.9
I feel that interleaving the tests with the code helps matching my skills to the challenge of the sub-problems I tackle on each iteration.	4.2	3.7
I feel that the burden of writing the tests as I write the actual code increases my feelings of anxiety and worriedness.	1.8	2.2

TABLE I
ADAPTED FLOW-STATE SCALE QUESTIONNAIRE RESULTS.

like I could control what I am doing". We also add a control question stating that the writing of tests before the production code actually increases anxiety.

We conducted an online survey with 86 volunteers from a tech forum specialized in discussions about TDD[1]. Among the respondents, 76% adopt TDD very often and were categorized as "experts" whereas 24% said that they adopt it only sometimes, being categorized as "intermediate" TDD practitioners. Results are summarized in Table I. Notice that the average flow score is high for all questions, ranging from 4.2 to 4.7 among experienced TDD practitioners, and from 3.6 to 4.0 among intermediate practitioners – except for the control question. Most interestingly, scores are higher among more experienced TDD practitioners, what we interpret as a compelling evidence that the TDD structure indeed favors the achievement of the flow state.

IV. Evidence #3: Self-reports on TDD and flow

Here, we resort to self-report observational data found on online tech blogs by quoting programmers testimonials referring to the mental experiences they had while applying TDD. We found these self-reports by googling "TDD" appended by flow-related feelings described in the flow-state scale questionnaire [38]. Despite being a weaker evidence subject to various biases, we believe they provide an extra, uncorrelated signal that our hypothesis holds.

Reports on Clear Goals.

> TDD makes me feel comfortable when coding because I already know where my code is leading and **what should I put on my code**. It makes my coding activities easy and **directed to my target. [50]**

Reports on Challenge-skill Balance.

> **Looking at big targets sometimes disheartens the programmers** because of involved complications. With TDD, every test instills the confidence of moving in right direction. Every test and successfully completed component marks the win **giving you clear picture of "what has been completed and what is remaining to be done?"** [51].

Reports on Unambiguous Feedback loops.

> Realizing the **short feedback loop** as the cornerstone of TDD was a real AHA moment for me [52].
>
> The streamlined TDD process helps the team members to improve their delivering capability because of **providing the immediate feedback** on the developed components. The **shorter feedback sharing loop** squeezes the turnaround period for the elimination of identified defects [51].

Reports on Sense of Control.

> The TDD rhythm results in **confidence** built of many successes (the tests) [53].
>
> If new code fails some tests, the programmer can **simply undo or revert rather than debug excessively** [54].

V. Conclusions and Future Plans

This work is a first effort to connect test-driven development with its impact on developers' flow state, a mental state that produces boosts in enjoyment and productivity. We (1) provided a theoretical connection between TDD and triggers of flow, (2) conducted a survey using a standard questionnaire adopted by flow state researchers and (3) showed self-reports from the Web that provide some anecdotal evidences to our hypothesis. We observed that TDD induces a structured learning process that tends to naturally align the difficulty of the problem with the programmer's current knowledge about it. Moreover, the tests written *before* production code build a feedback loop that provides clear goals to chase whilst makes the programmer feel in control at all times, conditions that are critical to induce flow experiences [20]–[22], [41].

In terms of research methodology, we want to conduct controlled experiments conjugating the flow-state scale survey with actual programming tasks solved by developers, as a means of strengthening a causal link between TDD and flow. At a broader scope, this research is an initial exploration

[1]https://groups.io/g/testdrivendevelopment

on how software engineering practices trigger or limit flow state experiences. Additionally, we want to investigate signs of flow experiences in other modern processes of software engineering, which are heavily tailored towards iterative and feedback-providing processes [55]. We also want to correlate software design choices with flow; for example, in code that follows the open-closed principle (design for extension rather than changing), do developers feel less anxious and a stronger sense of control when they evolve code by extending it rather than editing it? Also, think of the anxiety and stress a programmer can potentially face when trying to refactor a code without tests [3]; we also plan to observe that through flow state lens.

ACKNOWLEDGMENTS

We thank Stone Co. for providing the environment and challenges in which the idea of this research came to life.

REFERENCES

[1] K. Beck, *Test Driven Development: By Example*. Addison-Wesley Professional, 2002.

[2] S. Siddiqui, *Learning Test-Driven Development: A Polyglot Guide to Writing Uncluttered Code*. O'Reilly Media, 2021. [Online]. Available: https://books.google.com.br/books?id=qOJxzgEACAAJ

[3] R. Martin, *Clean Craftsmanship: Disciplines, Standards, and Ethics*.

[4] D. Janzen and H. Saiedian, "Does test-driven development really improve software design quality?" *IEEE Software*, vol. 25, no. 2, 2008.

[5] R. Jeffries and G. Melnik, "Guest editors' introduction: TDD–the art of fearless programming," *IEEE Software*, vol. 24, no. 3, 2007.

[6] I. Salman, A. T. Misirli, and N. Juristo, "Are students representatives of professionals in software engineering experiments?" in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*. IEEE Press, 2015.

[7] M. F. Aniche and M. A. Gerosa, "How the practice of TDD influences class design in object-oriented systems: Patterns of unit tests feedback," in *Proceedings of the 2012 26th Brazilian Symposium on Software Engineering*, 2012.

[8] M. Siniaalto and P. Abrahamsson, "A comparative case study on the impact of test-driven development on program design and test coverage," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 2007, pp. 275–284.

[9] H. Munir, M. Moayyed, and K. Petersen, "Considering rigor and relevance when evaluating test driven development: A systematic review," vol. 56, no. 4, 2014.

[10] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus, "What do we know about test-driven development?" *IEEE Software*, vol. 27, no. 6, 2010.

[11] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un) happy," *Journal of Systems and Software*, vol. 140.

[12] D. Graziotin, X. Wang, and P. Abrahamsson, "Happy software developers solve problems better: psychological measurements in empirical software engineering," *PeerJ*, vol. 2, p. e289, 2014.

[13] F. Fagerholm and J. Münch, "Developer experience: Concept and definition," in *2012 international conference on software and system process (ICSSP)*.

[14] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*. USA: Dorset House Publishing Co., Inc., 1987.

[15] M.-A. Storey, T. Zimmermann, C. Bird, J. Czerwonka, B. Murphy, and E. Kalliamvakou, "Towards a theory of software developer job satisfaction and perceived productivity," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, 2021.

[16] M. Csikszentmihalyi, *Flow: The psychology of optimal experience*. Harper & Row New York, vol. 1990.

[17] ——, *Finding Flow: The Psychology Of Engagement With Everyday Life*. Basic Books, 2020.

[18] D. Van der Linden, M. Tops, and A. B. Bakker, "The neuroscience of the flow state: involvement of the locus coeruleus norepinephrine system," *Frontiers in Psychology*, vol. 12.

[19] M. Csikszentmihalyi, *Beyond Boredom and Anxiety*, ser. The Jossey-Bass behavioral science series. Jossey-Bass Publishers, 1975.

[20] C. J. Fong, D. J. Zaleski, and J. K. Leach, "The challenge–skill balance and antecedents of flow: A meta-analytic investigation," *The Journal of Positive Psychology*, vol. 10, no. 5.

[21] D. Linden, A. Bakker, and M. Tops, "Go with the flow: A neuroscientific view on being fully engaged," *European Journal of Neuroscience*, vol. 53, 11 2020.

[22] D. J. Harris, S. J. Vine, and M. R. Wilson, "Neurocognitive mechanisms of the flow state," *Progress in brain research*, vol. 234.

[23] M. Ulrich, J. Keller, and G. Grön, "Neural signatures of experimentally induced flow experiences identified in a typical fmri block design with bold imaging," *Social cognitive and affective neuroscience*, vol. 11, no. 3.

[24] M. Ulrich, J. Keller, K. Hoenig, C. Waller, and G. Grön, "Neural correlates of experimentally induced flow experiences," *Neuroimage*, vol. 86.

[25] S. Kotler, *The Art of Impossible: A Peak Performance Primer*. HarperCollins, 2021. [Online]. Available: https://books.google.com.br/books?id=luTdDwAAQBAJ

[26] F. Stamatelopoulou, C. Pezirkianidis, E. Karakasidou, A. Lakioti, and A. Stalikas, "Being in the zone: A systematic review on the relationship of psychological correlates and the occurrence of flow experiences in sports' performance," *Psychology*, vol. 09, pp. 2011–2030, 01 2018.

[27] A. I. F. . W. M. Khoshnoud, S., "Peripheral-physiological and neural correlates of the flow experience while playing video games: a comprehensive review," *PeerJ*, 2020.

[28] K. Kiili, A. Perttula, A. Lindstedt, S. Arnab, and M. Suominen, "Flow experience as a quality measure in evaluating physically activating collaborative serious games," *International Journal of Serious Games*, vol. 1, 09 2014.

[29] M. Klasen, R. Weber, T. T. Kircher, K. A. Mathiak, and K. Mathiak, "Neural contributions to flow experience during video game playing," *Social cognitive and affective neuroscience*, vol. 7, no. 4.

[30] L. Harmat, Ö. de Manzano, and F. Ullén, *Flow in Music and Arts*. Cham: Springer International Publishing, 2021, pp. 377–391.

[31] M. Csikszentmihalyi and J. LeFevre, "Optimal experience in work and leisure." *Journal of personality and social psychology*, vol. 56, no. 5, p. 815, 1989.

[32] K. Kuusinen, H. Petrie, F. Fagerholm, and T. Mikkonen, "Flow, intrinsic motivation, and developer experience in software engineering," in *Agile Processes, in Software Engineering, and Extreme Programming*, H. Sharp and T. Hall, Eds. Cham: Springer International Publishing, 2016.

[33] K. Kuusinen, "Are software developers just users of development tools? assessing developer experience of a graphical user interface designer," in *Human-Centered and Error-Resilient Systems Development*. Springer International Publishing, 2016.

[34] R. Latorre, "Effects of developer experience on learning and applying unit test-driven development," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, 2014.

[35] M.-A. Storey, B. Houck, and T. Zimmermann, "How developers and managers define and trade productivity for quality," in *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 26–35. [Online]. Available: https://doi.org/10.1145/3528579.3529177

[36] J. Spacco and W. Pugh, "Helping students appreciate test-driven development (TDD)." New York, NY, USA: Association for Computing Machinery, 2006.

[37] S. Freeman and N. Pryce, *Growing Object-Oriented Software, Guided by Tests*, 1st ed. Addison-Wesley Professional, 2009.

[38] M. Kawabata, C. Mallett, and S. Jackson, "The flow state scale-2 and dispositional flow scale-2: Examination of factorial validity and reliability for japanese adults," *Psychology of Sport and Exercise*, vol. 9, pp. 465–485, 07 2008.

[39] A. Castillo-Rodríguez, U. Lopera, W. Onetti, and J. L. Minguet, "How and why do young soccer players change the flow state?" *PLOS ONE*, vol. 15, p. e0233002, 05 2020.

[40] S. Koehn and J. Díaz-Ocejo, "Imagery intervention to increase flow state: A single-case study with middle-distance runners in the state of qatar,"

International Journal of Sport and Exercise Psychology, pp. 1–14, 05 2016.

[41] A. Elliot and C. Dweck, *Handbook of Competence and Motivation*. Guilford Publications, 2005. [Online]. Available: https://books.google.com.br/books?id=B14TMHRtYBcC

[42] S. Kotler, M. Mannino, S. Kelso, and R. Huskey, "First few seconds for flow: A comprehensive proposal of the neurobiology and neurodynamics of state onset," *Neuroscience Biobehavioral Reviews*, vol. 143, 2022.

[43] G. B. Moskowitz and H. Grant, *The psychology of goals*. Guilford press, 2009.

[44] C. Alameda, D. Sanabria, and L. Ciria, "The brain in flow: a systematic review on the neural basis of the flow state," 11 2021.

[45] V. Khorikov, *Unit Testing Principles, Practices, and Patterns: Effective testing styles, patterns, and reliable automation for unit testing, mocking, and integration testing with examples in C#*. Manning, 2020.

[46] C. L. Bong, K. Fraser, and D. Oriot, *Cognitive Load and Stress in Simulation*. Cham: Springer International Publishing, 2016, pp. 3–17.

[47] J. Hirsh, R. Mar, and J. Peterson, "Psychological entropy: A framework for understanding uncertainty-related anxiety," *Psychological Review*, vol. 119, pp. 304–320, 12 2012.

[48] J. Schmidt, *Flow in Education*, 12 2010, pp. 605–611.

[49] S. Mentz, K. Owen, and T. Stankus, *99 Bottles of OOP - A Practical Guide to Object-Oriented Design*, 2nd ed., 2022.

[50] Test driven development is red-green-blue-development. [Online]. Available: https://medium.com/basic-people/test-driven-development-is-red-green-blue-development-f268e2150981

[51] 6 compelling benefits of (TDD) test driven development. [Online]. Available: https://www.knowledgehut.com/blog/agile/6-compelling-benefits-of-tdd-test-driven-development

[52] Does TDD slow you down or help you go faster? [Online]. Available: https://dev.to/toureholder/comment/1dfga

[53] Test driven development and flow. [Online]. Available: https://www.agilecoachjournal.com/2008-04-20/test-driven-development-and-flow

[54] Test driven development. [Online]. Available: https://www.goconqr.com/pt/fluxograma/29200361/test-driven-development

[55] D. Farley, *Modern Software Engineering: Doing What Works to Build Better Software Faster*. Addison-Wesley Professional, 2021.