External imagery in computer programming

Joseph Ditton
jditton.atomic@gmail.com
Utah State University
Logan, Utah

Hillary Swanson
hillary.swanson@usu.edu
Utah State University
Logan, Utah

John Edwards
john.edwards@usu.edu
Utah State University
Logan, Utah

ABSTRACT

Imagery is a cognitive process commonly used in sports in which athletes internally or externally visualize themselves performing a skill, allowing them to create an internal experience similar to the physical event. It is intended to allow participants to refine and perfect their performance. This paper investigates the use of imagery in the setting of computer programming. We explore the idea that watching a keystroke replay of yourself writing computer code that solves a specific problem can increase the speed and quality of a subsequent attempt at solving a similar problem as well as improve attitude and engagement. We investigate the theory of imagery, its application to computer programming, and we present results of a qualitative study. Our results suggest that using imagery could have a positive effect on the profitability of spending time reviewing code.

CCS CONCEPTS

• **Software and its engineering** → *Software design techniques*; *Collaboration in software development*; • **Social and professional topics** → *Computing education.*

KEYWORDS

student success, imagery, code review

ACM Reference Format:
Joseph Ditton, Hillary Swanson, and John Edwards. 2021-. External imagery in computer programming. In *The 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 17–20, 2021, Toronto, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Imagery is a technique used commonly in sports where the athlete rehearses their sport in real-time either in their mind or through some other visualization activity like a video recording [9]. This rehearsal is intended to both provide the athlete additional practice when physical practice isn't possible or convenient and also to allow the athlete to visualize attempts of skill performance that are successful. We suggest that some of the benefits of doing imagery could be applicable to computer science (CS) as a discipline, specifically in the area of code review.

Traditional code reviews where engineers read and discuss code are common in industry settings and occur occasionally in educational contexts. While useful, a student might be more advantaged by reviewing the *process* of writing their code, in what might be termed a "code development" review. Research has suggested that computer programmers, especially novices, think of their computer program less as a static product and more as the process they took to get to the final product [12, 13, 18]. In that sense, allowing a student to review their code by watching a replay of their keystrokes may be more effective as it may access their existing mental model of their program. During a static code review a student is required to reconstruct the process in order to identify areas of improvement, whereas using external imagery such as a keystroke playback sets the student in line with their existing mental representation and may allow them to more easily conceive changes to their process and design. Such reviews could be used to provide insight into a programmer's software development and problem solving processes with applications in computer science education as well as in professional development settings.

In this paper we report results from an empirical study where college students wrote a short computer program in an IDE that recorded timestamped keystrokes. Following their attempt, students in the test group were shown a playback of their typing and control group students were shown a static representation of their code. Following the code review session, all students wrote a second, similar computer program and then responded to a survey asking them about their experience. We report results of the survey to answer our research question, which is: *What attitudinal and engagement effects does external imagery have on students compared to students asked to review static code?* We find from our study that attitude and engagement appear to be better among students given an external imagery treatment compared to students asked to perform a static code review.

In this paper we present the theory behind imagery as well as prior work on code reviews in Section 2 followed by our methods in Section 3. We then present empirical results in Section 4 and discussion and conclusions in Section 5.

2 THEORY AND RELATED WORK

Performance review is a ubiquitous practice in all areas of expert performance. Two types of performance review are external imagery, typically used in sports and athletics, and code reviews, common in computer programming. This section discusses these two types of review, as well as theory on students' mental models that may indicate that imagery may be more suitable to performance improvement of novice programmers.

2.1 Imagery

External imagery is an activity common in sports where athletes watch video playbacks of themselves with the goal of self-evaluation and performance improvement [9]. It has been shown to improve athletic performance and has become more common with scientists recognizing the "psychological aspects of sport" [9]. Internal imagery is also a common practice, where athletes internally visualize themselves performing expertly, expecting performance improvements as a result. Scientific investigations into imagery have attempted to identify its functions. One of the earliest was that of Paivio [17], who proposed that imagery had influence on not only cognitive mechanisms (e.g., skill, strategy) but also on motivational mechanisms (e.g., self-confidence, motivation, goal-setting). Later work by Hall et al. [10] enumerated five main functions of imagery: 1) cognitive specific (e.g., skill); 2) cognitive general (e.g., strategy); 3) motivational specific (e.g., goal-setting); 4) motivational general-arousal (e.g., relaxation, anxiety, arousal); and 5) motivational general-mastery (e.g., self-confidence, focus, control). MacIntyre et al. [15] concluded that imagery also positively influences metacognition, which is unsurprising considering the linkages between metacognition and expertise [14].

It has been found that imagery is more effective in closed-skill sports than open-skill sports [1]. Closed-skill sports are those like golf or ice-skating, that involve performing a routine or repetitive action in a largely invariant environment. The more consistent the routine or action is the better the performance of the sport will be, as the primary motor cortex in the brain is stimulated when motion is imagined [19]. Other sports like wrestling or basketball are open-skill sports. While these sports have some closed skills, like shooting free-throws for example, the majority of the sport involves a broad range of often personalized skills that change their implementation based on game state and environment. Computer programming is a mix of open and closed skills. Closed skills include typing and syntax recall, whereas open skills include problem solving and debugging.

2.2 Code review and mental models

Peer code review is a common practice in software engineering settings, where programmers and engineers meet together to discuss the current state of a codebase. In professional settings, code reviews are designed primarily to detect and correct defects in code [3]. And, indeed, the primary benefits of code reviews have been code quality [4, 16], although soft-skills such as knowledge transfer and team awareness have also been found to improve [3]. Educators have found value in code reviews for pedagogical purposes, with improvements in student outcomes [20] as well as communication, teamwork, and attitudes [11].

However, novice computer programmers may see their programs less as a static product and more as a temporal process. In the context of debugging, Katz and Anderson [13] suggest that whether a programmer uses a forward- or backward-reasoning strategy when debugging is based on the programmer's mental model. Indeed, they found that a subject debugging their own code tended to use backward-reasoning strategies (searching backwards from the incorrect behavior of the program) while a subject debugging code written by someone else used forward-reasoning (building a mental representation of the program starting with reading the code). A programmer's mental model of code is important when interacting with it. While novice programmers develop their code line-by-line and read new code linearly [21] they do have knowledge of both how the algorithm works and their intentions [12, 13]. Using the terminology of Pennington [18], novices have "text structure knowledge" and "plan knowledge" of their code, suggesting a somewhat more temporal model of code than they would have of code written by someone else. This idea suggests that external imagery, or a temporal representation of the construction of the program, could better engage students and direct them to improved practices because the external representation naturally fits with the students' internal mental representation.

To our knowledge no prior work has been published in the CS education literature that proposes a "playback" model of computer program review for self-evaluation and improvement.

3 METHODS

3.1 Study Design

In accordance with our IRB protocol, we recruited students in an upper-level/graduate CS course at a mid-sized public university in the US. The data collection portion of the study was held during class as part of a unit on quantitative methods in empirical studies. Students had the option to consent to having their data used in downstream analysis. All 19 students opted in.

On the day of the experiment, participants were randomly split into two groups, a control group and a test group. Students in the test group were re-seated on one side of the room while the control group moved to the other side such that no student could see the screen of someone in the other group. The test group was given a Python programming challenge and had 15 minutes to complete it. Upon completion, each individual in the test group then watched a keystroke playback of themselves doing the challenge. The recording played at 4x speed. This allowed the recording to be completed within five minutes. Before the playback, students were instructed to identify areas where they might improve their design and code quality. Once the test group was done watching the playback, they attempted a similar programming challenge with the same constraints.

The control group was given the same programming challenges with the same constraints as the test group. However, upon completion of the first challenge the control group did not watch a recording of themselves. They were given five minutes to review a read-only view of their completed challenge. During this review time they were also instructed to identify areas where they might improve their design and code quality.

The first programming challenge was to design a Python class called `MicrowaveOven`. The class was to have methods such as `open_door` and `insert_food`. Students had to handle error cases such as trying to insert food when food was already in the oven. The second challenge was to write a class called `Elevator` with very similar requirements. Hidden unit tests measured completion of the tasks.

3.1.1 Programming Challenge Design. Our study is focused on the code-writing process rather than specifically problem solving. To that end, each of the programming challenges were designed to maximize the amount of code that would be written during the 15-minute time limit. While program design and problem solving were components of the challenge, they were de-emphasized. This resulted in programming challenges that were somewhat easier but took longer to implement.

In each of the programming challenges participants implemented a relatively simple, but complete software system that was modeled after a real-world object. First, they implemented a software representation of a microwave oven and second was an elevator control system. Neither of these examples are technically difficult, however, the time constraint forced participants to think critically at the same time as writing the code. We observed that participants took a few moments at the start of each challenge to read and understand the problem space.

The primary motivation behind doing two different challenges instead of repeating the same challenge is that a second attempt on any problem will be significantly easier than the first. Some participants, especially those in the test group, might have even memorized their solution after doing it once and simply regurgitated that solution for the second attempt. However, doing two separate challenges was not without risk either. They needed to be designed carefully enough that the difficulty was nearly identical between the two. After reviewing the survey results, we believe that the challenges were roughly equal in terms of difficulty.

We measured the correctness of student implementations through a series of unit tests and allowed the test output to guide their problem solving. For example, if the participant's solution allowed food to be inserted into the microwave while the door was closed, then a unit test would fail and notify them why the failure took place. This was done in hopes that after one or two test failures they would have a good idea of what kinds of things they needed to think about while completing the challenges without having to spend a lot of time reading instructions.

3.1.2 Survey. Following the completion of the second challenge both groups were given a survey. The survey consisted of four likert-style questions with five responses each (where "M" means "Microwave" and "E" means "Elevator"):

(1) Which challenge was harder?
    - M was much harder than E
    - M was somewhat harder than E
    - M and E were about the same difficulty
    - E was somewhat harder than M
    - E was much harder than M
(2) Which challenge was more enjoyable
    - M was much more enjoyable than E
    - ...
    - E was much more enjoyable than M
(3) How useful was the [playback/code review]
    - Not very useful
    - A little bit useful
    - Somewhat useful
    - Mostly useful
    - Extremely useful
(4) How engaging was the [playback/code review]
    - Not very engaging
    - ...
    - Extremely engaging

The final survey question was a free-response question asking them to describe the thoughts they had about the [playback/code review] time. The question was: "Tell us your thoughts about the [playback/code review]" where the review method presented to the student was in the brackets.

3.2 Coding of the free responses

Two of the authors conducted a grounded analysis [6] of the full set of responses to identify common themes. These themes were then refined as codes. The codebook is shown in Table 1.

The codebook was used to train two students to code the full set of responses. The coders then individually categorized each response. Following this, they compared results, discussed discrepancies, and finalized the codes given to each response. Through this social moderation process [8], 100 percent agreement was reached for all responses.

It is important to note that a response missing a particular code does not imply that the response should be viewed as expressing the opposite of that code. For example, if a response was not coded as "generally engaging" it should not be assumed that the participant thought the review time was not engaging. Rather, there was not specific language in their response that indicated that they found it engaging.

4 RESULTS

4.1 Performance

We first report quantitative results from the study. No statistical significance tests were done because of small sample size, but we report raw data for completeness. Table 2 reports the number of students to complete the challenges, time taken, and compilation data. More students completed the second challenge than the first in both treatments. This is not surprising, given that the second challenge was very similar to the first, and many students may have needed to warm back up to Python classes. The number of total compilations decreased for both groups, yet, interestingly, the percentage of compilations that resulted in an error stayed roughly the same, presumably because students were writing more code between compilations, but that the code had fewer errors per change.

4.2 Attitude and engagement

Our research question is: *What attitudinal and engagement effects does external imagery have on students compared to students asked to review static code?*

4.2.1 Attitude toward programming challenges. As mentioned earlier, we attempted to make the two programming challenges similar in terms of difficulty. We found that the participants generally felt the challenges were similar in difficulty (Figure 1a) with all participants responding that, at most, one challenge was "somewhat" harder than the other. How enjoyable one challenge was compared to the other was similarly unaffected by the difference in treatment: most students responded that the challenges were equally enjoyable (Figure 1b), with three students in the control group and two in

	Code Name	Description
	No Progress	Response mentions that participant did poorly so the review was not helpful
See/ Visualize	Errors	Response mentions that they saw or were able to visualize errors that they made
	Process	Response mentions that they were able to see or visualize their process
	Desire to Improve	Response mentions that they tried to improve or that review helped them to improve in some way
	Nothing New Learned	Response mentions that the review didn't show them anything they didn't already know
	Reinforced Knowledge	Response mentions that the review reminded them of or reinforced existing knowledge
	Typing Errors	Response mentions that they made typing errors
	Slow Down	Response mentions that they that they needed to slow down and think
General Attitude	Positive	Response has a generally positive attitude about the review
	Negative	Response has a generally negative attitude about the review
	Engaging	Response generally describes review as engaging
	Not Engaging	Response generally describes review as not engaging
	Wanted Direction	Response indicated that they wanted more guidance / direction on what to do during review
	Wanted Interaction	Response mentions that they wanted to be able to interact with review more
	Found Errors	Response mentions that they found errors

Table 1: Code book for survey free responses

the test group responding that the second challenge, the Elevator, was somewhat more enjoyable. This was surprising, as we would have expected more students to find the second challenge more enjoyable simply because they performed better.

4.2.2 Attitude toward the review session. Students were asked how useful they found the review session to be. Responses between the groups were almost identical, with 9/19 stating it wasn't useful at all and the remainder reporting at least "a little" utility. However, the engagement responses were quite different: while 6/9 in the control group found the static code analysis to not be engaging at all, only 2/10 found the playback review not engaging.

We looked into the two participants from the test group who didn't find the playback engaging at all. When coding the free response answers, which we will explore more in the next section, we found that one of these student's answers identified that they did so poorly on the first challenge that there was ultimately no value to be found in the review. Unfortunately, the other participant didn't write anything for their free response question so we cannot know how they felt about it. We reviewed the playback of this participant's first challenge and discovered that they, too, did so poorly on the first challenge that the playback would not have been interesting. This implies that some level of expertise, relative to the task, is required for the playback to be useful, corroborating existing research that shows that imagery is a more effective tool when used by experts [2].

	Finished		Time		Compiles		Error %	
	M	E	M	E	M	E	M	E
Control group	2/8	6/8	14.6	8.1	13	6.5	.16	.2
Test group	1/10	9/10	13.9	12.9	16.5	13	.14	.14

Table 2: Results from the study by treatment group. *M* and *E* are *Microwave* and *Elevator*, respectively. *Finished* is the number of students who finished the programming challenge and *Time* is the median number of minutes taken by those who finished. *Compiles* is the median number of compilations per student and *Error %* is the percentage of those compiles that yielded an error. One student in the control group was not included due to an error in the software.

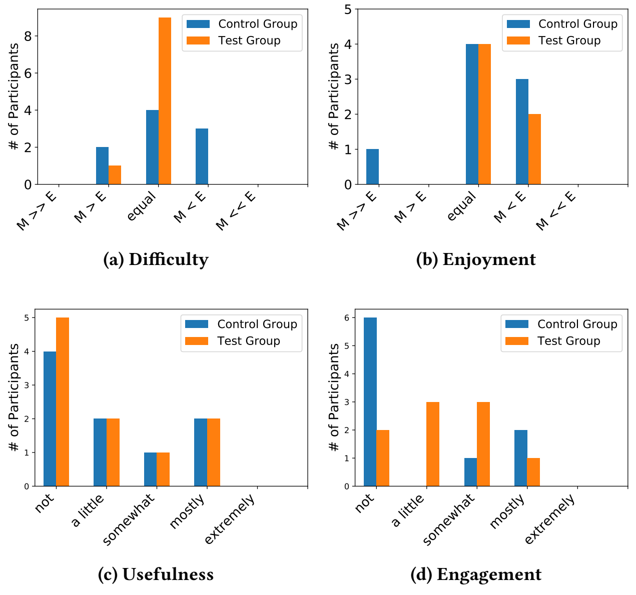

Figure 1: Likert-style response distributions.

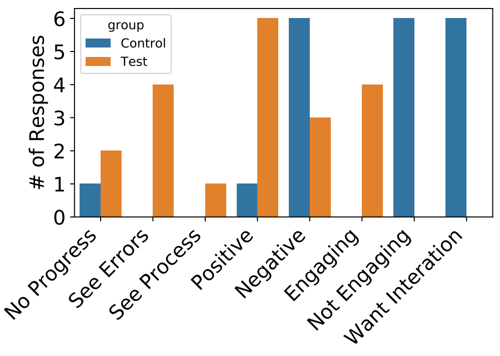

Figure 2: Coded free responses

4.2.3 *Free Response.* Seven control students and nine test students wrote responses to the free response question: *Tell us your thoughts about the [playback/code review].* Figure 2 shows coded responses. With only one exception, every response from the participants in the control group had a negative tone about the static code review. The only participant in the control group that had a generally positive response said the following:

> At first I thought it was pointless because I was not able to type any of my thoughts. After I got over my initial frustration about not being able to type I then looked through my code and found a few mistakes that I made sure I did not make in the next program.

Similar to the negativity toward the static code review, every participant but one in the control group also expressed that the review wasn't engaging. The remaining student didn't mention engagement at all.

Interaction was a theme among the control students. Every participant in the control group but one explicitly mentioned they wanted to interact in some way during the review. This response was typical of the others: "Without commenting, changing things, and running, I feel like reviewing it was pointless." Other than the one control student that made little or no progress on the first challenge, every response specifically mentioned one or more of these three things: commenting, changing, and running. Students wanted to actively interact with the code. This result is in alignment with previous research: the effect that student engagement has on learning effectiveness has been widely explored, and it is generally accepted that there is positive correlation between engagement and student success [5].

The "wanted direction" category is unique to the control group as well. Two of the control group participants specifically mentioned that they wanted some sort of guidance during the review. It is not surprising that no one in the test group expressed the same desire, since the playback acted as something of a step-by-step guide during the review time for the test group.

Responses from the test group regarding the playback review were of a different tone. No participant from the test group indicated that they did not find the playback engaging and four participants specifically indicated that they found the playback engaging.

The majority of the test group's responses were labeled as generally positive. The remaining three were labeled as generally negative. Two of these participants that reacted negatively to the playback were labeled as "no progress," meaning that they stated that they didn't make enough progress on the first challenge to make the playback review meaningful. The third student that reacted negatively simply felt the playback was useless.

Four of the nine responses specifically mentioned that the playback helped them see or visualize errors or process. One participant from the test group responded with the following:

> I liked being able to see my thought process as I worked through the problem. When I approached the Elevator problem, I tried to improve things I saw myself do that I could have done better.

No students given the static code review said anything about visualizing errors or process. This is an interesting result because the term "imagery" itself invokes a sense of visualization. While students in a static code review are, indeed, visualizing code, it is the students viewing the code as a playback that gain insights in a "visual" way.

5 DISCUSSION AND CONCLUSIONS

Our research question was: *What attitudinal and engagement effects does external imagery have on students compared to students asked to review static code?* Our results indicate a difference in attitude and engagement between the two treatments, with students appearing to prefer a playback review. However, while the free responses were fairly clear on engagement and usefulness, the likert-style questions were mixed, with perceptions of usefulness and effects on attitude toward the programming challenges roughly equal between the two treatments.

We motivated the study with theory backing external imagery in sports, but the analogy isn't perfect: in sports the only thing that makes sense to review is the action – reviewing the final score isn't useful. In contrast, computer programming has a sophisticated final product, a computer program, and in many situations a review of that product does make sense, such as in a group code review where engineers can deconstruct and discuss the code. The reviewers of the code don't already have a mental model of the code, and so constructing one during a review is reasonable. However, the *creator* of the code does have an existing mental model, and it is temporal. It is the process they went through to create the product. In this case, the sports analogy holds: it makes more sense for the programmer to review the performance itself, rather than the product of their performance.

Our results showed that students given the static code review almost universally, in both the likert-scale question and in the free response section, identified that they wanted more interaction during review time. Meanwhile, the test group thought the playback was generally engaging. Indeed, in addition to lack of engagement, the main theme of the control group appeared to be a desire for interaction. It is interesting that no test students indicated a desire for interaction, because the playback review was no more interactive than the static code review! It is possible that the engagement value supplanted the desire for interaction.

Control and test students were given the same amount of time to review their code. In a way, we might have expected the students given the static code review to be more successful because they were able to spend more time with the entire codebase, whereas students with the playback treatment didn't see the code in its entirety until just moments before starting the second challenge. The fact that students in the test group had a better experience and, from their free responses (though not necessarily from the likert-style questions), found more use out of the review is telling. It indicates that a review of the software development process may be perceived as more useful than a review of the source code itself.

There is a drawback to playback, and that is the fact that you can't get the whole view of the code at a glance – you have to wait for the playback to complete. Of course, interactive controls such as fast-forward, scrubbing, and jumping can make it easier, but it is still awkward at best, and could compromise the motivation for playback at worst.

For future work, we are interested in conducting a larger-scale study that includes quantitative analysis, as the results presented in this paper seem to indicate the potential of playback code review as a pedagogical tool. A feature of this second study could be allowing a group to continue working on the first challenge during the code review period. More broadly, the study presented in this paper explores external imagery and playback as a tool in computing education but, in a larger sense, it is advancing an investigation into mental representations of code that have a temporal element. We are interested in exploring these ideas further, including quantification of just how much of the process of writing the code is encoded in memory, what a temporal representation can mean for debugging techniques, and whether early CS students should be shown code being written rather than static code snippets.

5.1 Threats to validity

Most of our results center around self-reported engagement. It is possible that playback is more engaging simply because it is novel and dynamic. Also, the programming challenges were small, and so many students may have been able to find ways to improve their performance in just seconds of review rather than the five minutes given them, which would mean that the playback was simply entertainment. Another threat is that, while students in each group didn't know the treatment differences with the other group, students in the test group may have recognized the playback as the topic of interest and responded more positively than they would have normally in order to please the researchers [7].

REFERENCES

[1] Monna Arvinen-Barrow, Daniel A Weigand, Scott Thomas, Brian Hemmings, and Malcolm Walley. 2007. Elite and novice athletes' imagery use in open and closed sports. *Journal of Applied Sport Psychology* 19, 1 (2007), 93–104.

[2] Monna Arvinen-Barrow, Daniel A. Weigand, Scott Thomas, Brian Hemmings, and Malcom Walley. 2007. Elites and Novices Athletes' Imagery Use in Open and Closed sports. *Journal of Applied Sport Psychology* 19 (2007), 93–104.

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE).* IEEE, 712–721.

[4] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern code reviews in open-source projects: Which problems do they fix?. In *Proceedings of the 11th working conference on mining software repositories.* 202–211.

[5] Robert M. Carini, George D. Kuh, and Klein Stephen P. 2006. Student Engagement and Student Learning: Testing the Linkages. *Research in Higher Education* 47 (2006), 1–32.

[6] Juliet M Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology* 13, 1 (1990), 3–21.

[7] Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. 2012. "Yours is Better!": Participant Response Bias in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12).* Association for Computing Machinery, New York, NY, USA, 1321–1330. https://doi.org/10.1145/2207676.2208589

[8] John R Frederiksen, Mike Sipusic, Miriam Sherin, and Edward W Wolfe. 1998. Video portfolio assessment: Creating a framework for viewing the functions of teaching. *Educational Assessment* 5, 4 (1998), 225–297.

[9] Bruce D. Hale. 1994. *Imagery in sports and physical performance.* Baywood, Amityville, NY, Chapter 5, 75–96.

[10] Craig R Hall, Diane E Mack, Allan Paivio, and Heather A Hausenblas. 1998. Imagery use by athletes: development of the Sport Imagery Questionnaire. *International Journal of Sport Psychology* (1998).

[11] Christopher D Hundhausen, Anukrati Agrawal, and Pawan Agarwal. 2013. Talking about code: Integrating pedagogical code reviews into early computing courses. *ACM Transactions on Computing Education (TOCE)* 13, 3 (2013), 1–28.

[12] Elaine Kant and Allen Newell. 1984. Problem solving techniques for the design of algorithms. *Information Processing & Management* 20, 1-2 (1984), 97–118.

[13] Irvin R Katz and John R Anderson. 1987. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction* 3, 4 (1987), 351–399.

[14] Justin Kruger and David Dunning. 1999. Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of personality and social psychology* 77, 6 (1999), 1121.

[15] Tadhg E MacIntyre, Eric R Igou, Mark J Campbell, Aidan P Moran, and James Matthews. 2014. Metacognition and action: a new pathway to understanding social and cognitive aspects of expertise in sport. *Frontiers in Psychology* 5 (2014), 1155.

[16] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[17] Allan Paivio. 1985. Cognitive and motivational functions of imagery in human performance. *Canadian journal of applied sport sciences. Journal canadien des sciences appliquées au sport* 10, 4 (1985), 22S–28S.

[18] Nancy Pennington. 1987. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology* 19, 3 (1987), 295–341.

[19] Muriel Roth, Jean Decety, Monica Raybaudi, Raphael Massarelli, Chantal Delon-Martin, Christoph Segebarth, Stephani Morand, Angelo Gemignani, Michel Decorps, and Marc Jeannerod. 1996. Possible involvement of primary motor cortex in mentally simulated movement: a functional magnetic resonance imaging study. *Neuroreport* 7 (1996), 1280–1284.

[20] Saikrishna Sripada, Y Raghu Reddy, and Ashish Sureka. 2015. In support of peer code review and inspection in an undergraduate software engineering course. In *2015 IEEE 28th Conference on Software Engineering Education and Training.* IEEE, 3–6.

[21] Leon E Winslow. 1996. Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin* 28, 3 (1996), 17–22.