Using Games to Broaden Audiences for Programming Studies

Michael Coblenz
University of Maryland
College Park, Maryland, USA
mcoblenz@umd.edu

Felix Sosa
Harvard University
Cambridge, Massachusetts, USA
fsosa@fas.harvard.edu

ABSTRACT

Programming studies typically require participants with significant programming expertise. Because these experts are hard to recruit, programming studies can be challenging to run. Even when participants are recruited successfully, it is not clear how the results might relate to fundamental aspects of human cognition that drive participants' programming performance. We propose a new approach: map research questions onto specially-designed games, which could be played even by participants with limited programming background. This approach may lead to easier recruitment and more insight into focused questions about cognition.

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; • **Human-centered computing → HCI theory, concepts and models**; **Empirical studies in HCI**.

KEYWORDS

empirical studies of programmers, empirical software engineering, participant recruitment

ACM Reference Format:
Michael Coblenz and Felix Sosa. 2022. Using Games to Broaden Audiences for Programming Studies. In *Proceedings of 1st International Workshop on Recruiting Participants for Empirical Software Engineering (RoPES '22).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recruiting participants in studies of programming and software engineering can be challenging. Incentivizing professional software engineers with financial compensation is frequently beyond the budgets of academic labs, and some software engineers cannot write code for other organizations without their companies' consent. Recruiting students as participants can be feasible, depending on the required knowledge and skills, but results in limitations on external validity. Recruiting participants through online sources such as Mechanical Turk[1] (MTurk) can produce unreliable results [2, 10].

[1] www.mturk.com

RoPES '22, May/June, 2022, Pittsburgh, PA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Once recruited into a study, programmers frequently spend time on programming problems that are not of interest to the experimenter [4]. To mitigate this problem, study designers typically create tasks that minimize opportunities for these challenges both to reduce study length (and thus make recruitment easier) and to improve the signal-to-noise ratio for the variables of interest. The general approach is to conduct a variety of different studies and combine the results to better understand the relationship between design choices and programmer behavior.

Typical studies focus on evaluating specific designs. Unfortunately, these studies do not lead to direct insights about *human cognition*; due to the task complexity, it is usually impossible to isolate specific cognitive processes. However, insight about cognition would be valuable because it could enable researchers to build generalizable theories, reducing the empirical burden. Inspired by work in cognitive science and AI that re-casts questions about real-world learning into questions about learning in video-game domains [8, 11], we propose to re-cast questions about programming into questions about game-playing. The idea is to design a game that reflects the key cognitive aspects of a programming task, establishing a formal mapping that preserves the behaviors of interest across both domains. We hope to focus the participants' attention on a specific aspect of cognition, rather than requiring participants to do general programming tasks, and expand the participant pool from those with programming proficiency to anyone who can play the game. In doing so, we hope to obtain insights about cognition that could lead to theoretical development.

Reducing the required background for participants in programming studies may make recruiting substantially easier. In addition, by studying a larger pool, we may understand how individual cognitive abilities and limitations (e.g., limits on working memory capacity [1]) affect skill; and, by having a diverse group of participants, we may learn to what extent the results are generalizable across a variety of dimensions, such as proficiency and experience.

Two techniques can establish a correspondence between the two domains. First, we will show an isomorphism between key aspects of the problems that participants must solve. For example, a problem involving API calls with specific parameter and return types might be isomorphic to a particular jigsaw puzzle. The isomorphism can be made formal by deriving both the programming task and game task from the same programming language (e.g. the game can be within a visual programming language with the same semantics as the language in the programming task). Second, we will develop interventions that we hypothesize will affect behavior or performance, and then evaluate whether the interventions have similar effects in both contexts. For example, perhaps color-coding tabs of the puzzle or color-coding the type annotations in the textual language could help people more easily identify compatible and incompatible connectors. If color-coding improves performance in

both contexts similarly, this can provide evidence that the two tasks are related. In addition, we plan to also vary the difficulty of both the game and the programming task to see whether the difficulties of the two tasks are correlated.

Do results with games generalize to real programming tasks? We can provide support for a relationship between game designs and programming studies by, at first, doing *both* sets of studies. Prior to any empirical evaluation, if applicable, a formal mapping can be made between the design of the intervention of interest (e.g., a programming language construct) and a game mechanic. Provided the mapping is rigorous, this gives researchers a formal motivation for designing the study and exploring it empirically.

Task success can depend *both* on the underlying structure *and* on task presentation. A classic example is from Wason et al. [12]: participants were more accurate at identifying potential violations of a rule when it was presented in a real-world context than abstractly. Differences in behavior between corresponding contexts may provide insight as to how to make programmers more effective.

Another challenge is the immediate engineering costs–designing games can be difficult, and the mapping between the interventions a researcher might want to investigate and the game mechanics that those interventions map to might be difficult to establish. However, these costs can be mitigated by sticking to the kinds of "toy" games found in cognitive science, such as old Atari games, where the game mechanics are kept fairly simple but relevant [7].

When evaluating a correspondence between a game and a programming setting, it will be necessary to recruit equivalent participants across the two settings. Once the relationship has been established, however, the game study could be extended with participants who would not be prepared to do the programming study in order to see how the cognitive results might generalize to a broader population. The goal, however, is that once the principles of establishing relationships between programming problems and games are established, one can conduct studies only in the game setting. A challenge, then is to develop enough theory to show a close enough relationship between a game design and a programming context that conducting the programming study as well is unnecessary.

There is a possibility of a different level of engagement across the two settings, which could be controlled for by providing incentives for rapid completion. There is also a risk that the kinds of people who are attracted to the game study may be different from those who are attracted to the programming study. This could be controlled for with careful recruitment and by controlling for the demographic differences between groups.

2 EXAMPLES: MAPPING TASKS TO GAMES

Here, we give examples of mapping traditional programming-study tasks to games.

Studying cognitive load via a cooking game. The theory of working memory suggests that people have limited capacity storage for new information [1]. Cognitive load theory [9] predicts that people learn most effectively if the facts relevant to new information fit in short-term memory. This may have implications on programming tasks, which frequently require remembering relevant variables, functions, type definitions, and other facts [3].

If cognitive load theory is relevant to programmers, an empirical study could show how performance suffers when cognitive load is high. A study might ask programmers to do programming tasks under varying conditions of cognitive load (e.g., by changing the number of variables that must be used) and observing whether performance degrades when the number of relevant variables exceeds the participant's working memory limit.

Consider a game that simulates cooking. The player must combine ingredients together using various cooking processes, such as mixing or frying. Does performance (in errors made or time required) degrade when the size of the list of ingredients or processes exceeds the player's working memory limit, or if the processes become time dependent (e.g. player has to avoid burning food)? The study could attract participants more readily than the corresponding programming study, since it would not require any programming background. As a result, it would allow researchers to study impact of cognitive load on performance for a much broader population. A separate study could look at practicing programmers and assess whether (a) people with larger working memory capacities are more effective programmers; or perhaps (b) programming practice *increases* one's working memory capacity, allowing better performance on non-programming tasks.

Studying parallelization via a factory game. In SpaceChem [13], players construct factories to synthesize chemicals. These factories accept reagents and process them along player-designed pipelines to produce outputs. The pipelines operate in parallel, requiring users to think about problems of timing and synchronization. Prior studies of programmers regarding parallelism and concurrency [5] have been confounded by the specific training given to the participants. In contrast, games such as SpaceChem can be played with little training and could serve to provide evidence on the usability of parallel and concurrent programming constructs.

Abstraction. Some languages provide features that promote extensive use of polymorphism. For example, Haskell supports higher-kinded polymorphism, in which type variables can be of any kind (not only `Type`). What are the usability costs of such abstraction? Games, particularly ones that present puzzles, could represent the key structures of abstraction, and be used to understand the cognitive implications of highly abstract structures. Educators can use concrete examples to help students understand abstract principles [6]; understanding the relationship between the cognitive implications of abstraction and features that are provided in languages and libraries may aid in language and library design.

3 CONCLUSION

Recruiting participants in programming studies that produce high-quality data remains a challenge. However, re-casting problems in terms of games may broaden the pool of participants, enable researchers to obtain insights about cognitive processes that are relevant to programming, and enable more precise control over the problems the participants face. In doing so, we hope to build theories of cognition that reduce the need for high-cost empirical studies in language and tool design.

REFERENCES

[1] Alan Baddeley. 1992. Working Memory. *Science* 255, 5044 (1992), 556–559. https://doi.org/10.1126/science.1736359 arXiv:https://www.science.org/doi/pdf/10.1126/science.1736359

[2] Michael Chmielewski and Sarah Kucker. 2019. An MTurk Crisis? Shifts in Data Quality and the Impact on Study Results. *Social Psychological and Personality Science* 11 (10 2019), 194855061987514. https://doi.org/10.1177/1948550619875149

[3] Michael Coblenz. 2021. Toward a Theory of Programming Language and Reasoning Assistant Design: Minimizing Cognitive Load. arXiv:2110.03806 [cs.PL]

[4] Michael Coblenz, Gauri Kambhatla, Paulette Koronkevich, Jenna L. Wise, Celeste Barnaby, Joshua Sunshine, Jonathan Aldrich, and Brad A. Myers. 2021. PLIERS: A Process That Integrates User-Centered Methods into Programming Language Design. *ACM TOCHI* 28, 4, Article 28 (jul 2021), 53 pages. https://doi.org/10.1145/3452379

[5] Michael Coblenz, Robert Seacord, Brad Myers, Joshua Sunshine, and Jonathan Aldrich. 2015. A course-based usability analysis of Cilk Plus and OpenMP. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 245–249. https://doi.org/10.1109/VLHCC.2015.7357223

[6] Julie Colhoun, Dedre Gentner, and Jeffrey Loewenstein. 2008. Learning abstract principles through principle-case comparison. In *Proceedings of the 30th Annual Conference of the cognitive Science Society*. Cognitive Science Society, 1659–1664.

[7] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L Griffiths, and Alexei A Efros. 2018. Investigating human priors for playing video games. *arXiv preprint arXiv:1802.10217* (2018).

[8] Anna Raffert, Matei Zaharia, and Thomas Griffiths. 2012. Optimally designing games for cognitive science research. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 34.

[9] John Sweller and Paul Chandler. 1994. Why some material is difficult to learn. *Cognition and instruction* 12, 3 (1994), 185–233.

[10] Mohammad Tahaei and Kami Vaniea. 2022. Recruiting Participants With Programming Skills: A Comparison of Four Crowdsourcing Platforms and a CS Student Mailing List. *CHI Conference on Human Factors in Computing Systems* (2022).

[11] Pedro A Tsividis, Joao Loula, Jake Burga, Nathan Foss, Andres Campero, Thomas Pouncy, Samuel J Gershman, and Joshua B Tenenbaum. 2021. Human-level reinforcement learning through theory-based modeling, exploration, and planning. *arXiv preprint arXiv:2107.12544* (2021).

[12] P. C. Wason and Diana Shapiro. 1971. Natural and contrived experience in a reasoning problem. *Quarterly Journal of Experimental Psychology* 23, 1 (1971), 63–71. https://doi.org/10.1080/00335557143000068 arXiv:https://doi.org/10.1080/00335557143000068

[13] Zachtronics. 2012. *SpaceChem.* https://www.zachtronics.com/spacechem/