Reading Logic as Code or as Natural Language Text

A Randomized Controlled Trial Experiment on the Comprehensibility of Object-oriented Source Code in Comparison to Natural Language Text

Patrick Rein
patrick.rein@hpi.uni-potsdam.de
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany

CCS CONCEPTS

• **Software and its engineering → General programming languages**; *Collaboration in software development*; • **General and reference →** *Empirical studies.*

KEYWORDS

program comprehension, source code, natural language text, domain-specific languages, controlled experiment, Smalltalk

ACM Reference Format:
Patrick Rein. 2019. Reading Logic as Code or as Natural Language Text: A Randomized Controlled Trial Experiment on the Comprehensibility of Object-oriented Source Code in Comparison to Natural Language Text. In *Companion of the 3rd International Conference on Art, Science, and Engineering of Programming (Programming '19), April 1–4, 2019, Genova, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3328433.3328464

1 INTRODUCTION

Creating useful software systems requires a solid understanding of the application domain of the system [1, 9]. Software development teams thus often communicate with domain experts to elicit domain knowledge and requirements for the system [4]. To capture the resulting shared understanding of the domain, software developers and domain experts create artifacts such as graphical models or glossaries [5]. Source code is another interesting format for capturing the shared understanding of the domain as source code files are the definitive documents for the actual behavior of a software system.

At the same time, source code is a technical artifact and thus might be unsuitable for the communication between non-programming domain experts and the software development team. Domain-specific languages and graphical modeling languages are often proposed to enable domain experts to participate in the modification of source code documents [2, 6, 10]. These approaches assume that general-purpose programming languages result in source code documents that are not accessible enough for non-programmers to work with them.

Programming '19, April 1–4, 2019, Genova, Italy

ACM ISBN 978-1-4503-6257-3/19/04.
https://doi.org/10.1145/3328433.3328464

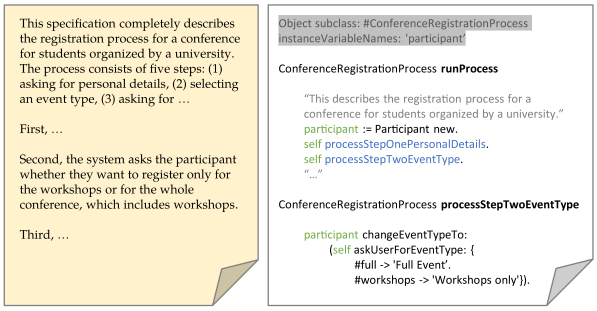

Figure 1: Sections from the English- and the Smalltalk-variant of one of the two scenarios including the original formatting.

We conducted a randomized controlled trial experiment to investigate this assumption. In detail, we investigated the hypothesis that:

Given a problem domain with simple rules, people with little to no programming experience score worse on a text comprehension task given a Smalltalk program in comparison to an English text document.

We chose the object-oriented paradigm, as it is used for describing domain models [4]. We chose Smalltalk [7] as the programming language because its syntax was designed to be minimal and resemble English text.

The insights from our study are related to the empirical evidence gathered on the effects of language features on programming novices [3, 12]. Further, our results also relate to results from studies investigating the effects of a domain-specific language on the productivity of experienced programmers [8]. However, in contrast to these studies, we focus on non-programming domain experts which do not intend to become proficient programmers.

2 EXPERIMENTAL DESIGN

We conducted a fixed-setup study through a 2x2 factorial experiment [11]. The first factor was the format of the document: Smalltalk source code or English text. To mitigate carry-over effects between rounds, we introduced two scenarios as the second factor (see Figure 1). For each scenario, we devised ten questions. We designed the scenarios to be of equal complexity to reduce the noise introduced by the scenarios.

We operationalized the comprehension level as the number of correctly answered questions on the content of the scenarios and

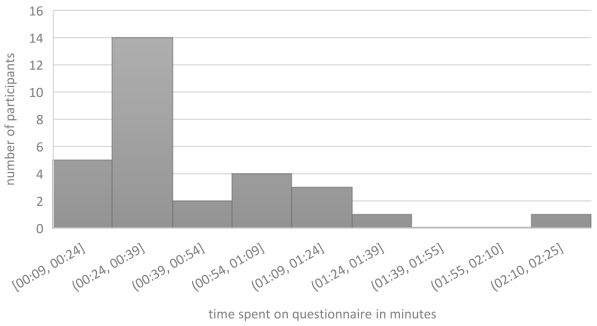

Figure 2: A histogram of the time spent working on the complete experiment in buckets of fifteen minutes.

the programming experience as the number of lines of code written in the past reported through formal self-evaluation.

We conducted the experiment through Amazon Mechanical Turk (MTurk). In total, we recruited 36 participants which had to hold a U.S. bachelor's degree. We ensured proper engagement with the task through control questions. After discarding submissions from participants with prior programming experience and submissions which answered any control question wrongly, we had 31 participants.

3 RESULTS

As we employed a within-subject design, we used a paired t-test for comparing the results (individual data see Figure 3 and Figure 4). Therefore, we determined the mean of the differences between the scores for the Smalltalk and the English language document of each participant. The assumption of normality was met. The mean difference is −1.42 (standard deviation 1.747) and the difference between the scores is significantly different from zero ($t_{31} = 4.524$, $p < 0.001$). This difference means that on average participants scored fewer points on questionnaires about a Smalltalk document than they did on questionnaires about a text document.

To check our assumption that the two different scenarios do not influence the results, we also analyzed the difference between the scores for the conference registration and the shop checkout scenario. Again, assumption of normality for the differences was met. The mean difference between the two scenarios is not significantly different from zero ($t_{31} = 1.656$, $p = 0.108$).

3.1 Threats to Validity

We identified the following major threats to the validity of the experiment. An external threat is the assumption that readers have no programming background. In practice this might not hold for domain experts working with software developers. An internal threat to validity is the little amount of time invested by participants (see Figure 2). Another internal threat is the potential difference in task difficulty as suggested by outliers in the data (see Figure 4).

4 CONCLUSION AND FUTURE WORK

The results of our experiment suggest that object-oriented source code written in Smalltalk is less comprehensible to non-programming domain experts for the tested types of scenarios. At the same time

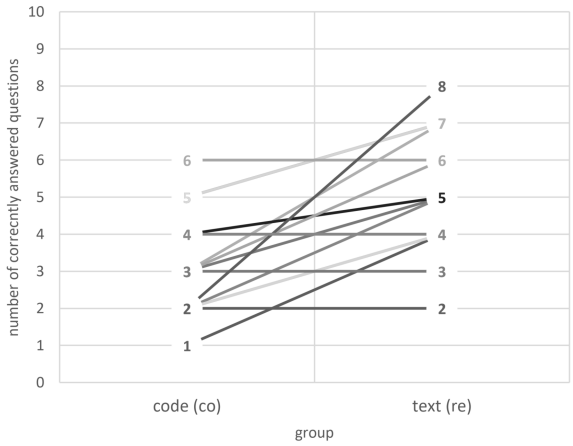

Figure 3: A chart showing the comprehension score of each participant (connected by a line). This chart shows all participants working on the checkout scenario in code and the registration scenario in text.

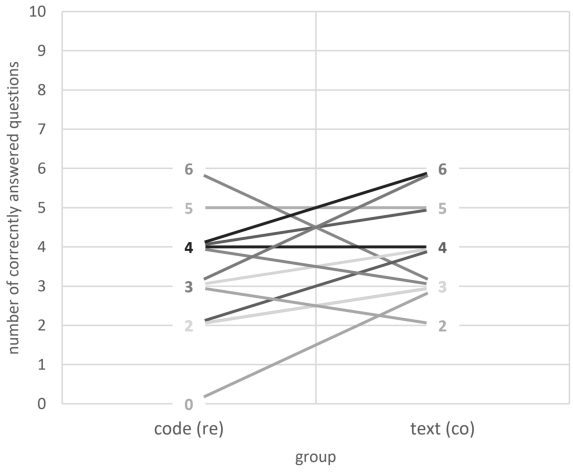

Figure 4: A chart showing the comprehension score of each participant (connected by a line). This chart shows the complementary group of participants to Figure 3: all participants working on the checkout scenario in text and the registration scenario in code.

the difference is less pronounced than we expected. Overall, the results should be regarded as preliminary, as outliers suggest remaining issues with the comparability of the scenario difficulty.

For future work, we will revise the experiment setup to eliminate any potential influence from the different scenarios so we can use the experiment as a baseline for further experiments into the comprehensibility of source code for non-programmers.

REFERENCES

[1] Kent Beck. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] T. Cheng, E. Lock, and N. Prywes. 1984. Use of Very High Level Languages and Program Generation by Management Professionals. *IEEE Transactions on Software Engineering* SE-10, 5 (1984), 552–563. https://doi.org/10.1109/TSE.1984.5010279

[3] Alireza Ebrahimi. 1994. Novice Programmer Errors: Language Constructs and Plan Composition. *International Journal of Human-Computer Studies* 41, 4 (1994), 457–457. https://doi.org/10.1006/ijhc.1994.1069

[4] Eric Evans. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley Professional.

[5] David N Ford and John D Sterman. 1998. Expert knowledge elicitation to improve formal and mental models. *System Dynamics Review: The Journal of the System Dynamics Society* 14, 4 (1998), 309–340.

[6] Martin Fowler. 2010. *Domain-Specific Languages.* Pearson Education.

[7] Adele Goldberg and David Robson. 1983. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[8] Johann Thor Mogensen Ingibergsson, Stefan Hanenberg, Joshua Sunshine, and Ulrik Pagh Schultz. 2018. Experience Report: Studying the Readability of a Domain Specific Language. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18).* ACM, New York, NY, USA, 2030–2033. https://doi.org/10.1145/3167132.3167436

[9] Gerald Kotonya and Ian Sommerville. 1998. *Requirements Engineering: Processes and Techniques* (1st ed.). Wiley Publishing.

[10] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and How to Develop Domain-Specific Languages. *Comput. Surveys* 37, 4 (2005), 316–344.

[11] Colin Robson. 2002. *Real World Research.* Blackwell Publishing.

[12] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *Trans. Comput. Educ.* 13, 4, Article 19 (Nov. 2013), 40 pages. https://doi.org/10.1145/2534973