Understanding Loop Iteration Effects on Program Comprehension

Calvin Josenhans

Indiana University

Abstract

Understanding how programmers comprehend code is important for improving software development efficiency and reducing cognitive load. In this study, we investigate how the number of loop iterations in a program affects code comprehension. We present a web-based experiment in which six experienced Python programmers were asked to predict the output of short programs involving iteration over lists. All programs were under 15 lines long and shared identical cyclomatic complexity, but varied in list length and in the use of additional state within the loop. Results show that increasing list length led to longer response times and lower accuracy, indicating a growing cost of mental simulation as the number of iterations increases. Programs requiring additional state tracking further reduced performance, suggesting increased working memory demands. We also observed significant learning effects across experimental blocks, with faster responses over time but no corresponding improvement in accuracy, due to increased speed of recognition of the program.

Keywords: Code Comprehension, Cognitive Science

Understanding Loop Iteration Effects on Program Comprehension

The development of computer programs is a significant and cognitively complex undertaking. Programmers use various control-flow constructs, such as loops and conditionals, as well as data structures such as lists and trees to write programs that cover a broad range of disciplines and use cases.

Software development consists both of writing code, and of understanding the code that others have written. It is not uncommon for many people to be working on the same parts of a code base and finding themselves needing to comprehend code in order to make their own additions to it. Understanding this code comprehension process and the factors that make it difficult allows for identification of areas where improvements can be made to ease the cognitive load of comprehension, thus saving both time and money.

We present a web-based experiment in which participants were asked to predict the output of several short programs featuring iteration over lists. The programs were all under 15 lines long and had the same cyclomatic complexity (McCabe, 1976), however they differed in the number of iterations of the loop that the subject had to simulate. We designed these programs to answer the question of how the number of loop iterations affects code comprehension, and to what extent that effect is dependent on additional factors of the program structure.

Related Work

Research into programmers' mental models has resulted in the development of various theories as to how programmers develop a mental model of a program upon first observation, in order to be able to understand, debug, and modify it. The bottom-up perspective posits that understanding starts with combining syntactic elements of source code into higher and higher levels of abstraction of understanding. However other theories propose that programmers have schema present for common program patterns and constructs, and the activation of these informs the programer's model of the program

(Heinonen et al., 2023).

Studies have shown that there are so-called "rules of discourse" (Soloway and Ehrlich, 1984) that describe conventions that programming experts have that effect their understanding of programs. Some of these are relevant for loop comprehension, such as the expectation that a looping construct such as `while` will only be used in cases where the body should be executed more than once. Research has also shown that in some cases expertise makes programmers more susceptible to certain kinds of errors where they have not been trained (Hansen et al., 2013), and that whitespace is relevant for judging whether statements are judged as being chunked together as part of the same loop body.

Research has also been devoted to understanding code complexity metrics, and how well they correspond to or predict human comprehension. Models of mental simulation process are developed, based on human subject performance at simulating containing loops with a limited number of iterations (Nakamura et al., 2003), where in this case the goal was to avoid effects of "loop induction" by the participants. This found a significant mental cost incurred by non-constant variables. An fMRI study comparing various code complexity metrics measured subject brain activity while they simulated a program, some of which contained loops (Peitek et al., 2021). Here similarly program vocabulary size was seen to burden working memory.

Method

Subjects

The participants in this study were six students at Indiana university, all of whom self reported as having at least three years of experience with the Python programming language. Additionally, all self-described as being either advanced or intermediate Python programmers, with three falling into each category.

Apparatus

The studied was executed through the use of an online experiment created with the jsPsych framework[1]. Participants completed the experiment on their own computers at a time of their own discretion.

Design

Two independent variables were manipulated within each subject. **List length** was either 1, 5, 10, or 20, and describes the length of list in the code sample that was being iterated over. **Program type** describes the type of task being carried out by the iteration in the program. Three categories of programs were determined: `nostate` programs consisted counting of occurrences of simple arithmetic properties in a list; `count_state` programs used an additional variable to count certain pairings of numbers in a list, such as a number occuring twice in a row. `hard` programs used both an additional state variable, and used the variable of loop operation for comparisons within the loop. All programs shared the same cyclomatic complexity. All code fragments used in the experiment can be found in our GitHub repository. [2]

For each trial, the dependent variables measured were the response time, and whether or not the given response was correct.

Procedure

Three blocks of conditions were created by combining base programs with randomly generated lists of a particular length. Within each block no programs were repeated, but each program occurred once per block, and would appear in a random order. The program was presented on the screen with instructions to predict its output, which the subject could type in a text box, and submit their final response. One second of wait time was given

[1] https://www.jspsych.org/latest/

[2] https://github.com/HalflingHelper/code-comprehension

between each program, and subjects could pause for as long as they wanted between each of the three blocks.

Results

Data analysis was performed in R. Any regressions were done with the built in `lm` function, and ANOVAs were performed with the `ez` package's `ezANOVA` function.

Accuracy

Subject accuracy was measured as the percentage of correct responses to the programming problems. Out of 216 total problems answered by our subjects, 66 of them, or 30.6%, were given as incorrect answers. We note, however, that 77.3% of these were so-called "off-by-one" errors, as seen in figure 1.

Figure 1

Distribution of errors

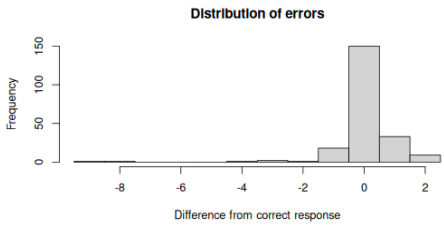
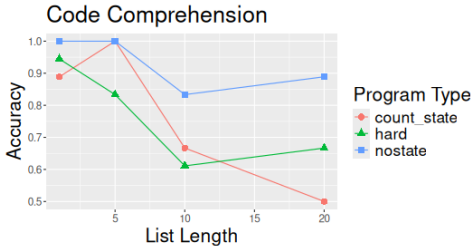


Figure 2

Accuracy effects



We found a main effect of both code type ($p = 0.01$) and list length ($p = 0.002$) on accuracy. Accuracy decreased for code fragments with longer lists, and accuracy was lower for the `count_state` and `hard` program types. Though a statistically significant interaction between code type and list length was not found ($p = .11$), we do see a greater difference between the programs for longer lists.

Response Time

For analysis of response times we only considered correct responses to the program prompts.

We found main effects of program type ($p = 0.045$) and list length (intercept $= 16324, \text{coef} = 1125, p < 0.05$) on response time. The `nostate` programs had significantly quicker response times than either of the other two program types, and as expected longer lists had longer response times, as there was more data for the subjects to process.

As with accuracy, we did not find a significant interaction between program type and list length ($p = 0.42$).

Figure 3

Program type and Response time

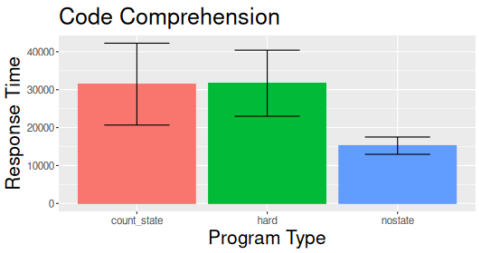
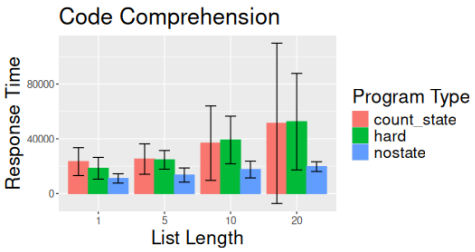


Figure 4

List length and Response time



Learning Effects

We found a significant effect of block index on reaction time (intercept $= 35802$, $\text{coef} = -10338$, $p < 0.05$), indicating that subjects improved at the task over the course of the observation. However there was no correlation between block index on subject accuracy, so while speed increased, it did not necessarily result in improved performance.

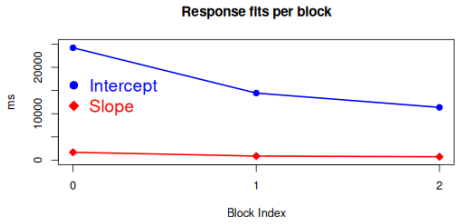
For regressions of response time on list length, the intercepts stayed consistent across the three blocks, while the slope decreased.

Discussion

Our results show that increasing the number of iterations in a program increases the time taken for mental simulation of the program, indicating that to at least some extent, the programmer is taking time to manually progress through loop iterations.

Figure 5

Slopes and Intercepts



We observed significant learning effects on the tasks, and see that improvements in response time are due to improved recognition of the particular program, rather than proceeding faster through each iteration of the loop. This indicates that the subjects are perhaps building schema of the particular programs we used here, even over the short course of the experiment.

With regards to accuracy at the task, we see incidents of incorrect answer increase with list length, and no correlation between response time and accuracy. This suggests that mistakes are not a result of subjects spending more time on the code as a whole, but due to errors in the operations they are simulating, and even as that simulation speed increases, there is no corresponding improvement in accuracy. The error rates were higher as well in programs with more variables being used, so as more working memory is occupied, errors are more likely to happen.

We were unable to determine whether the relationship between list length and response time was a strictly linear one, or whether more complex behavior was at play, due to both limited participants, and a limited selection of list lengths observed.

Future Work

The results here show promise for probing deeper into how programmers understand loop execution. Further study might incorporate eye-tracking or fMRI measurements as other code comprehension investigations have to draw physiological grounding for our

observations here. There is also a much greater variety of program structure available to probe, and perhaps loops in the context of larger programs are understood differently. Additionally, though subjects here were all fairly experienced programmers, investigation into whether the learning effects we observed remain present in novices would also offer insight into how programmers learn to adapt.

References

Hansen, M. E., Goldstone, R. L., & Lumsdaine, A. (2013). What makes code hard to understand? *arXiv (Cornell University)*. https://doi.org/https://doi.org/10.48550/arxiv.1304.5257

Heinonen, A., Lehtelä, B., Hellas, A., & Fagerholm, F. (2023). Synthesizing research on programmers' mental models of programs, tasks and concepts — a systematic literature review. *Information and Software Technology*, *164*, 107300. https://doi.org/https://doi.org/10.1016/j.infsof.2023.107300

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Nakamura, M., Monden, A., Itoh, T., Matsumoto, K.-i., Kanzaki, Y., & Satoh, H. (2003). Queue-based cost evaluation of mental simulation process in program comprehension. *Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)*, 351–360. https://doi.org/https://doi.org/10.1109/METRIC.2003.1232480

Peitek, N., Apel, S., Parnin, C., Brechmann, A., & Siegmund, J. (2021). Program comprehension and code complexity metrics: An fmri study. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 524–536. https://doi.org/https://doi.org/10.1109/ICSE43902.2021.00056

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595–609. https://doi.org/https://doi.org/10.1109/tse.1984.5010283