Title

Investigating the Relationship Between Surprisal and Processing in Programming Languages

Journal

Proceedings of the Annual Meeting of the Cognitive Science Society, 46(0)

Authors

Dodd, Nicole
Reese, Skyler Jove
Morgan, Emily

Publication Date

2024

Peer reviewed

Investigating the Relationship Between Surprisal and Processing in Programming Languages

Nicole Dodd (ncdodd@ucdavis.edu)
Department of Linguistics, University of California, Davis, Davis, CA 95616

Skyler Reese (mwreese@ucdavis.edu)
Department of Linguistics, University of California, Davis, Davis, CA 95616

Emily Morgan (eimorgan@ucdavis.edu)
Department of Linguistics, University of California, Davis, Davis, CA 95616

Abstract

This study explores the relationship between predictability, as measured by surprisal, and processing difficulty in code comprehension. We investigate whether similar mechanisms govern the processing of programming and natural languages. Previous research suggests that programmers prefer and produce more predictable code, akin to natural language patterns. We utilize eye-tracking data from the Eye Movements in Programming (EMIP) dataset to examine the impact of surprisal on various eye movement measures. Contrary to expectations based on natural language processing, our results reveal that surprisal does not significantly influence fixation metrics. Additionally, regressions in code reading show an unexpected inverse relationship with surprisal, suggesting that readers have different reasons for making regressions while reading code versus natural text. These findings contribute insights into the unique dynamics of code comprehension and opens avenues for further research in this domain.

Keywords: language processing; programming languages; eye tracking and reading; predictive processing

Introduction

An emerging area of interest in the field of language processing is the processing of programming languages, or code. Programming has primarily been viewed through the lens of logic and problem-solving, and the "language" component is often overlooked. With the increasing prevalence of programming, including proposals to recognize programming languages as a type of "foreign language" in secondary education, researchers have recognized the need to further explore the cognitive processes involved in reading and comprehending code (e.g., Fedorenko et al., 2019). A key question in this discussion has been, are the cognitive mechanisms recruited while processing programming languages similar to those used for processing natural languages?

Comparing Natural and Programming Languages

Natural and programming languages share many similarities that may entail similar processing mechanisms. For example, we often use similar language to describe the use of natural and programming languages: we speak of "reading and writing" code, there are well-known programming language "idioms" that can be language-specific, and many languages have "coding style" guides to maintain writing consistency by language. Like natural languages, programming languages are symbolic systems of communication that allow for creativity in language use. On a concrete level, both natural and programming languages have lexicons from which they can draw to build meaningful phrases and are constrained by the grammatical rules of syntax.

However, there are also intrinsic differences between the two that may lead to distinct processing mechanisms. For one, most individuals would agree that programming languages are decidedly not natural languages. Humans cannot claim to be "native" speakers of programming languages, and the use of programming languages is constrained to very domain-specific environments. Programming languages also differ from natural languages in their syntax and lexicon: programming languages have infinitely large lexicons as programmers can constantly make up new function and identifier names, but its syntax is far more limited than that of natural languages.

Existing literature investigating the underlying cognitive mechanisms of code processing is limited in scope and provides conflicting evidence. On the one hand, there is some evidence that programming languages are processed in a similar manner as natural languages. For example, corpus studies show that programmers intentionally write predictable code (Casalnuovo et al., 2019), and behavioral studies find that predictable code improves comprehension times (e.g., Casalnuovo et al., 2020a; Hansen et al., 2013), similar to natural language production and comprehension. Further, some neuroimaging studies suggest a considerable overlap in the areas of the brain recruited for natural and programming language processing (Siegmund et al., 2014; 2017). On the other hand, there is evidence that programming languages are processed in a distinct manner from natural languages. For one, programming languages have also been found to be more cognitively demanding to read and write than natural languages (Busjahn et al., 2011). Programmers also read code less-linearly than natural language (Busjahn et al., 2015); however, this difference in behavior is dependent on the expertise of the programmer (e.g., advanced programmers have even less linear reading patterns and are more likely to skip intermediate words and lines). Other neuroimaging studies contra those mentioned above also

In L. K. Samuelson, S. L. Frank, M. Toneva, A. Mackey, & E. Hazeltine (Eds.), *Proceedings of the 46th Annual Conference of the Cognitive Science Society.*

found that distinct areas of the brain are activated while reading natural versus programming languages (Ivanova et al., 2020), and that a domain-general system in the brain was recruited for programming language processing instead.

This work aims to further investigate similarities and differences in processing code and natural language by examining the relationship between predictability and processing difficulty in code. We are specifically interested in determining how expectations, as measured by surprisal, affect processing behaviors in code, and whether surprisal has a similar effect on processing with code as it does on natural language.

Predictability and Language Processing

We know that code is more cognitively demanding to process than natural language (Busjahn et al., 2011). One way that humans attenuate cognitive difficulty in processing natural language is by using more predictable language, or language that is more expected in context. Numerous psycholinguistic studies have demonstrated that the frequency and distributional properties of linguistic units, such as words, phrases, and syntactic structures, substantially influence how humans process language. For instance, words that are more frequent and predictable are read faster (e.g., Ehrlich & Rayner, 1981; Inhoff & Rayner, 1986), and reading syntactic structures that are more common and more predictable in context facilitates processing (e.g., Hale, 2001; Levy, 2008).

Probabilistic accounts of language processing have operationalized this relationship between expectations and processing difficulty through surprisal, which is the negative log probability of a word in context. For example, in self-paced reading tasks, longer reading times are directly correlated with higher surprisal, or low predictability (e.g., in the sentence "I took my dog to the …", "park" would be a low surprisal option while "circus" would be a higher surprisal option) (Smith & Levy, 2013). In eye-tracking studies, this difficulty can present as longer total fixation times, increased regressive saccades, or both (Frazier & Rayner, 1982; Staub, 2010).

Researchers once wondered whether programming languages were more predictable than natural languages and found that they indeed were. Hindle et al. (2012) set out to test whether code or natural text was less predictable by training predictive language models on code and natural text corpora. They found that language models trained on code exhibited lower cross-entropy values than those trained on text. In light of these findings, researchers explored whether the observed predictability in code resulted from programmers reducing cognitive load by opting for predictability amidst multiple options, or was merely a natural side effect of code's grammatical and syntactic constraints. Casalnuovo et al. (2019) investigated this question by conducting a corpus analysis comparing the predictability of 5 programming languages to 3 natural languages. To control for code's limited grammar and syntax, the authors focused their analyses on open-class words and compared code corpora to natural language corpora of technical writing, which warrants repetitive and predictable language in a similar way as code. They trained various language models on these corpora and found that programming languages had consistently smaller entropy values, and were thus more predictable than natural language, even when accounting for grammatical and syntactic constraints. The authors conclude that while some of the difference in predictability is due to grammatical constraints, a significant portion must also be due to human preferences for predictable code.

Casalnuovo et al. (2020b) then decided to explicitly test programmers' preferences for predictable code. They conducted an experiment where programmers were presented with two versions of an expression and asked to select their preferred version. One version of the expression was with a canonical ordering (assumed to be preferred), and the second was with a non-canonical ordering derived through a meaning-preserving transformation. For example, the expression i + 1 can be expressed as either i + 1 or 1 + i and be meaningfully equivalent; however, i + 1 is the preferred and more frequent ordering. The authors also trained language models on these alternations to quantify which alternation was more predictable (i.e., had lower surprisal). They found that participants were more likely to choose variants with lower surprisal ratings as the preferred alternation, thus supporting their previous conclusion that humans do prefer, and likely write, predictable code.

Predictable code has also been shown to facilitate code processing and comprehension, similar to natural language. For example, Hansen et al. (2013) conducted an eye tracking study in which programmers were presented with functions that contained predictable and less predictable alternations and asked to summarize the purpose of the function. They found that less predictable code caused longer response times when determining the purpose of the function. Notably, in certain cases, this effect declined with experience, such that programmer experience helped to counteract the effect of seeing less predictable code. The authors interpreted this to be due to real-world experience with code that is both predictable and unpredictable, so expectations from real-world experience superseded baseline preferences.

Since surprisal has been shown to be strongly correlated with processing difficulty in natural language, a natural question would be whether this correlation also exists with processing difficulty in programming languages. Casalnuovo et al. (2020a) tested whether surprisal is correlated with code processing difficulty through a human behavioral study. They found a significant effect of surprisal on time spent answering comprehension questions about functions, and a non-significant trend towards a negative relationship between surprisal and comprehension question accuracy.

Current Study

Previous studies on code processing and comprehension have demonstrated that humans prefer predictable code (Casalnuovo et al., 2020b; Casalnuovo et al., 2020a), and predictable code generally facilitates processing times for

comprehension questions (Hansen et al., 2013); however, no studies have yet taken a granular look at how difficulty in code processing presents during incremental, on-line processing.

Our study takes steps toward filling this gap by identifying patterns of processing difficulty at the token level and asking whether surprisal is a significant predictor of these behaviors. We test our predictions by analyzing an existing eye-tracking corpus for code, where participants read two medium length code blocks and completed comprehension tasks. In natural language, surprisal is a predictor of processing difficulty, such that high surprisal values for a word would result in increased processing difficulty. In practice, this manifests as longer reading times or an increased proportion of regressive saccades away from the word into an earlier part of the sentence. We test whether this pattern holds with code by determining whether surprisal has any significant effect on processing behaviors while reading code. Further, we test whether surprisal affects similar reading measures in code as in natural language – while surprisal affects both early measures (e.g. first pass duration) and late measures (e.g. total fixation duration) for natural text, it is possible that the non-linearity of code would result in less effects on early processing measures (i.e., first pass) than overall processing measures. It is important to note that this preliminary analysis is limited by the nature of the pre-existing dataset; however, the insights will be a valuable first step toward investigating the relationship between surprisal and processing in programming languages.

Methods

Materials and Participants

The Eye Movements in Programming (EMIP) dataset (Bednarik et al., 2020) includes eye movements for 216 programmers (41 women; mean age: 27, *SD* = 9) of different experience levels completing two code reading and comprehension tasks, each comprising 11-22 lines of code (Figure 1). Data were collected by eleven research teams across eight countries. Participants were offered the option to complete the experiment in Java (~96% of participants selected), Scala (~2%), or Python (~2%). Prior to the start of the experiment, participants completed a demographic survey where they reported their years of expertise in programming (*M* = 6.0, *SD* = 7.9), their level of expertise in programming (*none*: 11, *low*: 45, *medium*: 120, or *high*: 40), their years of experience with the experiment language (e.g., Java) (*M* = 2.3, *SD* = 3.3), and their level of expertise with the experiment language (*none*: 30, *low*: 69, *medium*: 100, or *high*: 17). Participants were instructed to read each stimulus for comprehension and answered a multiple-choice question about the output of the function. Eye movements were recorded using a screen-mounted SMI RED250 mobile eye tracker with a sampling rate of 250 Hz. No head or chin rest was used in order to simulate a naturalistic programming environment.

```
public class Rectangle {
      private int x1 , y1 , x2 , y2 ;
      public Rectangle ( int x1 , int y1 , int x2 , int y2 ) {
            this.x1 = x1 ;
            this.y1 = y1 ;
            this.x2 = x2 ;
            this.y2 = y2 ;
      }
      public int width ( ) { return this.x2 - this.x1 ; }
      public int height ( ) { return this.y2 - this.y1 ; }
      public double area ( ) { return this.width ( ) * this.height ( ) ; }
      public static void main ( String [ ] args ) {
            Rectangle rect1 = new Rectangle ( 0 , 0 , 10 , 10 ) ;
            System.out.printIn ( rect1.area ( ) ) ;
            Rectangle rect2 = new Rectangle ( 5 , 5 , 10 , 10 ) ;
            System.out.printIn ( rect2.area ( ) ) ;
      }
}
```

Figure 1: Sample stimulus in Java from the EMIP dataset. Extra spaces were included between tokens to allow for character-level precision for fixations on punctuation and operators.

We used a filtered and corrected dataset, executed and published by the original authors, for our analysis (Al Madi et al., 2021). The corrected dataset includes token-level fixation information for all Java trials. Tokens were defined as being split by whitespace, and this definition was set by the original authors (Figure 1). Fixations that presented a clear shifting pattern were hand-corrected by one of two authors, who applied a general offset correction using the EMIP Toolkit in Python (Al Madi et al., 2021), and trials that were not fixed by the offset correction were excluded. Additionally, 15% of the data is not published.[1] We further filtered the dataset to exclude any participants who indicated that they had "none" programming experience (*N* = 11). The final dataset used for our analysis included 253 trials from 147 participants.

Participant Programming Expertise

Participant programming expertise was determined based on information provided in the demographic questionnaire. Each participant answered four questions about their programming expertise, including years of experience and their level of expertise for both programming in general and the experiment language. We decided to forgo using the measure based on years of experience as the distribution of answers was very broad and skewed (experiment language expertise: *M* = 2.3, *Mdn* = 1.0, *SD* = 3.3; programming expertise: *M* = 6.0, *Mdn* = 3.0, *SD* = 7.9). When then compared levels of expertise for general programming versus the experiment language and noticed a general pattern of participants rating themselves as less proficient in the experiment language than

[1] http://www.emipws.org/dataset/

Table 1: Raw averages for each eye tracking metric overall, by class, and by programmer expertise.

Metric	Overall	Open Class	Closed Class	Punct	Low	Med	High
First fixation duration	117	118	116	115	116	118	125
First pass duration	187	209	170	148	186	187	201
Total fixation duration	443	561	328	235	443	440	461
First pass regressions	0.11	0.09	0.09	0.23	0.11	0.11	0.11
Overall regressions	0.24	0.23	0.16	0.36	0.23	0.25	0.24
Overall skip rate	0.54	0.32	0.41	0.75	0.53	0.54	0.56

in programming overall (experiment language expertise – none: 30, low: 69, medium: 100, high: 17; programming expertise – none: 11, low: 45, medium: 120, high: 40). To avoid over-estimating the expertise of the participants, we used the level of expertise with the experiment language as our measure of participant programming expertise in our analyses.

Token Surprisal and Class

Surprisal values for each token in the stimuli were calculated using OpenAI's Codex (Chen et al., 2021). Codex is a GPT-3 based language model trained on both natural language and source code and is proficient in over a dozen programming languages. Due to internal tokenization protocols, transformer-based models like GPT-3 will often generate surprisal values for sub-word units. To align surprisal values generated from Codex to full tokens, we summed surprisal values within each token to generate a token-level value.

To control for differences in fixations on content (open class) versus function (closed class and punctuation) tokens, we assigned each token to one of the three classes. Following Casalnuovo et al., (2019), we used the type categorization feature from the Pygments[2] syntax highlighting library in Python to determine the class of each token. Token class was then used as a main predictor variable in our analyses described below.

Eye Movement Measures

The corrected EMIP dataset reported fixation durations and timestamps for individual token-level fixations for each trial. Using this information, we calculated the following eye movement measures for our analysis: *first fixation duration* (the duration of the first fixation on a token), *first pass duration* (the sum of all first pass fixations before leaving a token for the first time), *total fixation duration* (the sum of all fixations on a token), *first pass regression* (a binary measure indicating whether the reader's first pass through a token ended with a regressive saccade to an earlier part of the function), *overall regression* (a binary measure indicating whether any fixation on a token ended with a regressive saccade to an earlier part of the function, including first pass regressions), and o*verall skip rate* (a binary measure indicating whether a token was not fixated on for the duration of the trial). We chose to exclude certain metrics, such as first pass skip rates and go-past time, due to non-linear reading patterns with code (Busjahn et al., 2015).

Analysis and Results

The goal of our analysis was to investigate the relationship between surprisal and processing difficulty while reading code. We first calculated summary statistics to observe overall trends in the data. Raw metrics from the dataset are reported in Table 1.

The averages from this dataset show overall longer fixation times and increased regressive saccades compared to averages from natural text (Rayner, 1998), consistent with previous work in this area (Busjahn et al., 2011; Busjahn et al., 2015). Research on reading natural English text reports an average of 200-400 ms for total fixation durations and 10-15% rate for proportion of fixations that result in a regressive saccade (Rayner, 1998). The average total fixation duration in the EMIP dataset was 443 ms, which is higher than the averages for natural text, and 24% of fixations resulted in a regressive saccade, a higher proportion than in natural text. These data support previous claims that code is more difficult to process, and thus results in longer total fixation times and increased regressive saccades (Busjahn et al., 2011).

The averages further show sizable differences by token class. For example, the average total fixation duration for open class tokens was 561 ms (*SD* = 694) but only 328 ms (*SD* = 288) for closed class tokens. However, it is important to note that these averages do not control for token length, and open class tokens are on average longer than closed class tokens in this dataset (6.0 and 5.3 characters respectively). We also see substantially more regressions on open (23%) than closed class items (16%), but fewer skips on open (32%) than closed class items (41%).

Previous work has reported differences in eye movement behavior between novices and experts (e.g., Busjahn et al., 2015). We hypothesize that this is due to the relative simplicity of the stimuli used in the experiment and discuss this in further detail below.

To test the statistical significance of the effect of surprisal on these metrics, we modeled eye movement measures with mixed-effects regression models using the *brms* package in R

[2] https://pygments.org/

(Bürkner, 2018). Linear regressions were fit for fixation metrics and logistic regressions were fit for binary measures. We fit individual models for each eye movement metric. The models included sum-coded fixed effects for Token Class (open: 1, 0; closed: 0, 1; punctuation: -1, -1), Surprisal (continuous, centered) and Programming Language Expertise (high: 1, 0; low: 0, 1; medium: -1, -1), including all possible interactions. We also included control predictors by Token Length (continuous) and Regressed In (no: 1, yes: -1), indicating whether the first fixation on a token was the result of a regressive saccade from later in the stimulus. We used the maximal random effects structure justified by the design (Barr et al., 2013), resulting in random intercepts for Token ID and Participant, and random slopes by Programming Language Expertise for Token ID, and by Token Class and Surprisal and their interaction for Participant. We additionally used weakly informative priors on the Intercept and all main fixed effects. We consider the model estimates as reliable if the credible interval (CrI) does not include 0, or over 95% of the sampled posterior distribution is over or under 0 in the predicted direction. Model estimates for significant main effects and interactions are reported in Tables 2-3.

Table 2: Significant linear regression model estimates for fixation metrics.

Model Parameters	β	CrI
First fixation duration		
Class (open) * Expertise (high)	-3.58	[-7.29, 0.17]
Class (open) * Expertise (low)	2.01	[-2.65, 79.61]
First pass duration		
Class (open)	9.40	[1.29, 17.39]
Class (closed)	-9.74	[-19.89, 0.55]
Total fixation duration		
Class (open)	37.06	[-5.24, 78.79]
Class (closed)	-67.56	[-119.12, -16.39]

First Fixation Duration

We found no significant main effects for first fixation duration; however, we did find two significant interactions between Token Class and Programming Language Expertise. High expertise participants read open class tokens significantly faster while low expertise participants read open class tokens significantly slower. There were no significant main effects or interactions for surprisal.

First Pass

We found significant main effects for Token Class for first pass reading times, such that open class tokens are read slower while closed class tokens are read faster. There were again no significant main effects or interactions for surprisal.

Total Fixation Duration

We also found significant main effects for Token Class for total fixation durations, such that open class items are read slower while closed class tokens are read faster. There were again no significant main effects or interactions for surprisal.

Table 3: Significant logistic regression model estimates for regression and skip metrics.

Model Parameters	β	CrI
First pass regression		
Class (closed)	-0.47	[-0.66, -0.30]
Surprisal	-0.09	[-0.15, -0.02]
Class (open) * Expertise (low)	-0.07	[-0.16, 0.01]
Overall regression		
Class (open)	0.15	[0.04, 0.27]
Class (closed)	0.09	[-0.87, -0.53]
Surprisal	0.03	[-0.16, -0.04]
Class (open) * Expertise (high)	0.11	[0.00, 0.22]
Class (closed) * Expertise (high)	-0.21	[-0.41, -0.02]
Class (open) * Expertise (low)	-0.06	[-0.14, 0.01]
Class (closed) * Expertise (low)	0.12	[0.01, 0.24]
Overall skip		
Class (open)	-0.68	[-0.91, -0.45]
Expertise (low)	-0.19	[-0.37, -0.02]
Class (closed) * Surprisal	0.18	[0.02, 0.35]
Class (open) * Expertise (high)	-0.14	[-0.27, -0.01]
Class (closed) * Expertise (high)	0.31	[0.10, 0.52]
Class (open) * Expertise (low)	0.12	[0.04, 0.21]
Class (closed) * Expertise (low)	-0.34	[-0.48, -0.20]
Surprisal * Expertise (high)	-0.08	[-0.13, -0.02]
Surprisal * Expertise (low)	0.04	[0.01, 0.08]

First Pass Regressions

We found significant main effects for Token Class and Surprisal for first pass regressions, as well as a significant interaction between Token Class and Programming Language Expertise. Closed class tokens had significantly fewer first pass regressions, and higher surprisal values resulted in fewer first pass regressions. Low expertise participants also made significantly fewer first pass regressions on open class items.

Overall Regressions

We again found significant main effects for Token Class and Surprisal for overall regressions, as well as four significant interactions between Token Class and Programming Language Expertise. Open class tokens had significantly more regressions overall while closed class tokens had significantly fewer regressions overall. Surprisal also showed

a similar trend to first pass regressions, where higher surprisal values resulted in fewer first pass regressions. Interactions between Token Class and Programming Language Expertise showed contrasting patterns: high expertise participants made more regressions on open class tokens than closed class tokens, while low expertise participants made fewer regressions on open class tokens than closed class tokens.

Skips

We found significant main effects for Token Class and Programming Language Expertise for overall skips, as well as multiple significant interactions between Token Class and Surprisal, Token Class and Programming Language Expertise, and Surprisal and Programming Language Expertise. Open class tokens were significantly less likely to be skipped, and low expertise participants skipped significantly fewer tokens. While we found no main effects for Surprisal, there were numerous significant interactions with surprisal: for one, closed class tokens with higher surprisal values were more likely to be skipped. Further, tokens with higher surprisal values were less likely to be skipped by high expertise participants, but more likely to be skipped by low expertise participants. Finally, there were significant interactions between Token Class and Programming Language Expertise at multiple levels. High expertise participants made fewer skips on open class tokens and more skips on closed class tokens, while low expertise participants made more skips on open class tokens and fewer skips on closed class tokens.

Discussion

Our analysis takes first steps toward further investigating the relationship between surprisal and measures of processing difficulty in code in order to determine possible similarities or differences between human natural and programming language processing. We use an existing eye tracking dataset of participants completing two code comprehension tasks to complete our analysis. We examined the effect of token class type, surprisal, and programmer expertise on standard eye tracking measures, with special attention to measures typically associated with processing difficulty in natural language: total fixation durations and proportion of regressive saccades.

Our results showed no significant effect for surprisal on any of the fixation metrics gathered; rather, the main determiner of difference in reading times was token class, and to some extent, programmer expertise. Open class tokens were read longer while closed class tokens were read shorter, suggesting that programmers pay more attention to items like variable names than repeated, template-like function terms. This differs from what we would expect to see in natural language, where high surprisal items would cause significantly longer total durations.

Conversely, we do find a significant effect for surprisal on both regression measures; however, the effect is in the opposite predicted direction as we would see in natural language. In this dataset, tokens with high surprisal resulted in significantly fewer first pass and overall regressions away from the token. One aspect of reading code that we have previously discussed is that scan paths and fixation patterns are far less linear in code than in natural text (Busjahn et al., 2011). It has been hypothesized that regressions occur in reading natural text in order to re-analyze the previously processed input upon reading something difficult or unexpected (Altmann et al., 1992; Rayner, 1998); however, if code reading is non-linear, it's possible that regressions are made for reasons other than due to processing difficulty. It is also possible that readers need to read a few more progressive tokens to contextualize the high-surprisal item before regressing away to read something else. For example, if a reader encountered "1 + i" instead of the preferred "i + 1", they may continue past "1" to read the operator and variable before regressing away from the whole statement. We leave the further testing of these possibilities to future research.

Limitations and Future Directions

Our analysis provides a valuable and low-cost first look at the relationship between surprisal and processing difficulty in code. Our findings have identified potential key differences in how surprisal affects natural language versus code and highlighted methodological considerations for future work in this area. However, it is important to note that this analysis has some limitations. Primarily, some of the methodological choices for the original data collection affected our ability to conduct a thorough analysis. For one, the stimuli used were relatively short and straightforward, resulting in little variation in surprisal values in tokens, and thus made identifying a strong surprisal effect more difficult. Experimenters also did not use a head mount to stabilize participant movements, meaning the precision of the measurements reported in the data was perhaps not precise enough to confidently analyze token-level fixations. The EMIP is the only dataset of its kind to our knowledge, so we were limited in our ability to use data more specifically suited to our research question. Future research could use longer and more difficulty stimuli, providing more variation in surprisal values, as well as a higher precision eye tracker.

Acknowledgements

This research was supported by the National Science Foundation CCF (SHF-MEDIUM) Grant to EM [grant number 2107592].

References

Al Madi, N., Guarnera, D., Sharif, B., & Maletic, J. (2021). EMIP Toolkit: A Python Library for Customized Post-processing of the Eye Movements in Programming Dataset. *Eye Tracking Research and Applications Symposium (ETRA)*, *PartF169257*.

Altmann, G. T. M., Garnham, A., & Dennis, Y. (1992). Avoiding the garden path: Eye movements in context. *Journal of Memory and Language*, *31*(5), 685–712.

Barr, D. J. (2013). Random effects structure for testing interactions in linear mixed-effects models. *Frontiers in Psychology*, *4:328*.

Bednarik, R., Busjahn, T., Gibaldi, A., Ahadi, A., Bielikova, M., Crosby, M., Essig, K., Fagerholm, F., Jbara, A., Lister, R., Orlov, P., Paterson, J., Sharif, B., Sirkiä, T., Stelovsky, J., Tvarozek, J., Vrzakova, H., & van der Linde, I. (2020). EMIP: The eye movements in programming dataset. *Science of Computer Programming*, *198*.

Bürkner, P. C. (2018). Advanced Bayesian multilevel modeling with the R package brms. *R Journal*, *10*(1), 395–411.

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., & Tamm, S. (2015). Eye Movements in Code Reading: Relaxing the Linear Order. *2015 IEEE 23rd International Conference on Program Comprehension*, 255–265.

Busjahn, T., Schulte, C., & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, 1–9.

Casalnuovo, C., Devanbu, P., & Morgan, E. (2020a). Does Surprisal Predict Code Comprehension Difficulty? *Proceedings of the 42nd Annual Meeting of the Cognitive Science Society*, 564–570.

Casalnuovo, C., Lee, K., Wang, H., Devanbu, P., & Morgan, E. (2020b). Do Programmers Prefer Predictable Expressions in Code? *Cognitive Science*, *44*(12).

Casalnuovo, C., Sagae, K., & Devanbu, P. (2019). Studying the Difference Between Natural and Programming Language Corpora. *Empirical Software Engineering*, *24*, 1823–1868.

Ehrlich, S. F., & Rayner, K. (1981). Contextual effects on word perception and eye movements during reading. *Journal of Verbal Learning and Verbal Behavior*, *20*(6), 641–655.

Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (2019). The Language of Programming: A Cognitive Perspective. *Trends in Cognitive Sciences*, *23*(7), 525–528.

Frazier, L., & Rayner, K. (1982). Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, *14*(2), 178–210.

Hale, J. (2001). A probabilistic earley parser as a psycholinguistic model. *Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies 2001- NAACL '01*, 1–8.

Hansen, M., Goldstone, R. L., & Lumsdaine, A. (2013). What Makes Code Hard to Understand? *arXiv Preprint arXiv:1304.5257*.

Hindle, A., Barr, E. T., Gabel, M., Su, Z., & Devanbu, P. (2012). On the Naturalness of Software. *In Proceedings of the 34th International Conference on Software Engineering*, *59*(5), 837–847.

Inhoff, A. W., & Rayner, K. (1986). Parafoveal word processing during eye fixations in reading: Effects of word frequency. *Perception & Psychophysics*, *40*(6), 431–439.

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'Reilly, U. M., Bers, M. U., & Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive brain regions. *eLife*, *9*, 1–24.

Levy, R. (2008). Expectation-based syntactic comprehension. *Cognition*, *106*(3), 1126–1177.

Rayner, K. (1998). Eye Movements in Reading and Information Processing: 20 Years of Research. *Psychological Bulletin*, *124*(3).

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., & Brechmann, A. (2014). Understanding understanding source code with functional magnetic resonance imaging. *Proceedings of the 36th International Conference on Software Engineering*, *1*, 378–389.

Siegmund, J., Peitek, N., Parnin, C., Apel, S., Hofmeister, J., Kästner, C., Begel, A., Bethmann, A., & Brechmann, A. (2017). Measuring neural efficiency of program comprehension. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 140–150.

Smith, N. J., & Levy, R. (2013). The effect of word predictability on reading time is logarithmic. *Cognition*, *128*(3), 302–319.

Staub, A. (2010). Eye movements and processing difficulty in object relative clauses. *Cognition*, *116*(1), 71–86.