Searching for the Neural Correlates of Code

Michael E. Hansen

Indiana University

1 Specific Aims

Research into the psychological foundations of computer programming has been on-going since the 1950's [4]. Great strides have been made, from the simple act of equating complex software with basic structural features of the code [15, 12] to the use of complex models of text understanding that posit multiple levels of representation in the programmer's head [7]. Many behavioral studies of have been conducted on programmers in pursuit of an empirically-grounded cognitive model for program understanding [6]. To date, however, the neural correlates of program understanding remain a mystery.

It has been shown that programming experts, like experts in other domains (e.g., chess, physics, electronic circuitry), owe much of their expertise to the concordance of their memory **chunks** with the functional units of the domain [20]. Thus, expert programmers are able to efficiently encode the problem at hand and see the "bigger picture" behind the code whereas novices rely heavily on surface-level features of code to infer its meaning. The particular form of these chunks (or schemas) is still being debated, as is the way they are activated in response to a given task. The lack of conclusive behavioral results has given way to a plethora of theories that seek to explain the underlying mechanisms of program understanding.

One specific debate in the program comprehension literature centers around whether programmers represent chunks visually, semantically, or both. Based on the fact that programmers can recall code statements verbatim, Shneiderman and Mayer have proposed that programmers retain only the meanings of code statements (i.e., their semantics), and not their syntax [19]. In contrast, Parnin believes that the syntax of code statements is also retained during program comprehension via abstracted perceptual patterns [17]. According to text comprehension theories of programming, the semantic representation of code statements is linguistic (i.e., propositional) in nature [23]. Thus, we can potentially differentiate between chunk representations (semantic or syntactic + semantic) by associating them with well-known functional regions of the brain (language or visual working memory + language).

We propose an fMRI experiment that will be used to gather evidence for one of the preceding chunk representations (semantic or syntactic + semantic). We will require participants to copy snippets of code verbatim (called the **code copy task**) while in an MRI scanner under one of two conditions: (1) code is presented as written, and (2) code is presented with the lines randomly permuted. This is somewhat similar to early code recall tasks [20, 8], but is based more heavily on a chessboard copying task from early research with chess experts [2]. To our knowledege, this will be the first time that this kind of task has been performed with programmers in an MRI scanner. For this experiment, our **specific aims** are:

1. Between more and less experienced programmers, do we see a difference in activation within linguistic and visual working memory areas?

2. When code is scrambled by line, does this difference effectively disappear?

2 Significance

This experiment is a significant step forward in the study of program comprehension and expert memory. The production of software is one of the most complex activities that humans engage in every day, yet virtually nothing is known about its neurological basis. Studying the underpinnings of program comprehension is especially interesting, given that it involves visual, spatial, linguistic, and mathematical processes. Thus, the field is inherently inter-disciplinary.

Beyond a purely scientific interest, this experiment will also help to inform education in Computer Science. In a 2007 paper [14], Kramer looks back on 30 years of teaching and asks:

> *Why is it that some software engineers and computer scientists are able to produce clear, elegant designs and programs, while others cannot? Is it purely a matter of intelligence? Is it possible to improve the skills and abilities of those less able through further education and training?*

Answering these questions will require knowing **how** software engineers and computer scientists actually represent and construct programs. Due to contradictory behavioral results, the field of program comprehension needs studies such as this in order to provide clear constraints on potential models. This experiment is meant as a first step towards a comprehensive model of program comprehension that will span the perceptual and conceptual levels of code understanding.

3 Innovation

The experiment described here is innovative, since it combines methodologies from the fields of program comprehension and expert memory in order to create a new cross-disciplinary experimental task (the **code copy task**). Behavioral program comprehension studies to date have mostly involved program recall, fill-in-the-blank, or verbalization while problem solving. We present a simplified task that is modelled after Chase and Simon's classic experiment with chess masters [2].

Our simple code copy task will allow us to isolate the underlying representation of memory chunks in programmers by only requiring that they hold code in memory (and not manipulate it). Unlike a standard code recall task, the programmer is not trying to memorize an entire program at once and is not limited in the number of times they may edit their answer (except by a time limit – see Section 4.3). We believe the code copy task itself is innovative, acting as a bridge between Computer Science and the empirical brain sciences.

In addition to being cross-disciplinary, this is (to our knowledge) the first time that a program comprehension task will be performed in an MRI scanner. Previous work in program comprehension has focused exclusively on behavioral results, with a great deal of hypothesizing about the underlying brain mechanisms [17]. Our experiment employs state-of-the-art fMRI scanning techniques common in Psychology and Cognitive Science to help drive knowledge and research in Computer Science.

4 Approach

4.1 Background

Our experimental design closely mirrors that of Chase and Simon (1973) [2]. In their **chessboard copy task**, Chase and Simon had players copy the positions of every piece from one chessboard to another as quickly as possible. The two boards were placed such that they could not simultaneously be in view, ensuring that the player would need to quickly memorize as much as they could before moving over to the other board. In order to estimate the *chunk capacity* of short-term memory, Chase and Simon operationally defined a single chunk as a sequence of chess pieces placed by a player within two seconds of each other. In other words, it was assumed that two pieces placed in rapid succession (< 2s) were both present in the same memory chunk. With this definition, it appeared that experts and novices both shared the same short-term memory capacity (about seven chunks), but that experts stored more information per chunk than novices (i.e., experts placed more pieces per chunk on average).

In a later study of chess players (1996), deGroot and Gobet attempted to correct for a potential issue with the Chase and Simon study: players are physically only able to hold a small number of pieces in their hands at once. Given Chase and Simon's operational definition of chunk (pieces placed within 2s of each other), it was possible that experts were storing more than a (literal) handful of pieces in a single chunk; it might simply have taken more than two seconds to place them on the board. Using a computerized version of the chessboard copy task, deGroot and Gobet showed that Chase and Simon had overestimated the short-term memory capacity of their players. The new estimate was around four chunks for both experts and novices (compared to about seven chunks previously). Further research has lowered the estimated capacity of short-term memory even more, down to as little as two chunks [9]!

Given the limitations of short-term memory, deGroot and Gobet suggested that chess experts' performance is the result of having a vast library of information-rich chunks available in long-term memory. These chunks contain perceptual information about particular chess piece configurations, and are linked together to form what's called a discrimination network. A similar picture for computer programmers has been proposed by Gobet and Oliver, with code patterns playing the role of chess piece configurations [10]. For our experiment, we accept Gobet and Olver's hypothesis and assume that a programmer's code chunks are present in working memory when they are mentally representing code. Assuming this will allow us to use relative brain activation in particular areas to determine the neural representation of these code chunks.

4.2 Participants

We will recruit healthy, monolingual (English-speaking), right-handed participants who may or may not have any programming experience. All participants must be able to touch type on a standard QWERTY keyboard, which will be present in the scanner. We define a participant's "programming experience" as the number of years that they have worked with at least one programming language. Following Barfield et al. [1], we will divide participants into the following expertise groups: (1)

naïve participants, who have no programming experience, (2) novices, who have less than a year of experience, (3) intermediates, who have less than three years of experience, and (4) experts, who have more than three years of experience. The number of participants in each expertise group should be roughly the same in order to allow for equal comparisons.

Participants who have prior programming experience must have worked with a programming language which is present in our bank of sample programs (described in Section 4.3). If code chunks do contain syntactic (visual) information, then it is important that the right programming language be used. For our pilot study, these languages will include Java, Python, and C++. Participants without any programming experience must be native English speakers to ensure that the copy task can be done quickly enough (most programming languages include English keywords and identifiers). We expect naïve participants to perform much of the copy task through verbal rehearsal.

4.3 Methods

Our experimental setup will have participants lying in the MRI scanner with a computer screen and standard QWERTY keyboard. All keystrokes will be recorded and timestamped to allow for registration between the MRI and behavioral data streams. Participants will be continually scanned throughout each trial (defined below).

4.3.1 The Code Copy Task

We define the **code copy task** as follows:

1. For each trial (10 total, 5 trials per condition, randomly ordered), participants are presented with a screen of code (code screen) that they must copy **verbatim** into an empty text window (copy screen).

2. A hotkey is provided to switch between the code screen and copy screen such that only one can be active at a time (the clipboard will be disabled to avoid copy-paste).

3. Participants are informed that there will be a delay when switching from the code screen to the copy screen, during which the screen will be blank (there will not be a delay going from the copy screen to the code screen).

4. Participants are presented with a program drawn at random from our code bank (details below). In the **normal condition**, this program will be presented as written. In the **scrambled condition**, the program's lines will be randomly permuted (retaining indentation – see Section 4.4.1).

5. Once they believe that the program has been successfully copied, the participant will press a predefined key to continue to the next trial (they will be told to keep trying if there are mistakes – see Section 4.4.4 for concerns with this approach).

6. Participants will have up to 5 minutes per trial with an inter-trial interval of 15 seconds (a blank screen will be shown). If the 5 minute time limit is reached without a successful copy, the trial will automatically end and the next trial will begin (after the inter-trial delay).

From the description above of the code copy task, there are at least two important variables to consider: the **set of programs to copy** and the **switching delay** between the code and copy screens. The set of programs must fit several criteria: (1) they must be readable (i.e., formatted properly, not obfuscated), (2) they cannot be too simple or too complex (i.e., not a collection of simple print statements or the solution to a very obscure problem), and (3) they cannot be canonical (i.e., a well-known example from a book). These criteria are meant to ensure that experts are able to chunk some, but not all, of the code that they view under the normal condition. Under the scrambled condition, the expertise effect is expected to all but disappear (see next section).

Our program bank will be built using snippets of code from Google Code [3], a popular online code repository. This repository contains code from a wide variety of open source projects using different programming languages. For all languages in our pilot study, code snippets will be gathered by hand from the top five most popular projects on Google Code for that language. Hand-selecting snippets will ensure two things: (1) that each code snippet matches the criteria above, and (2) that each code snippet can be reasonably copied in under five minutes by a naïve participant. A study of average computer users has found that transcription speeds are around 33 words per minute [13], meaning that snippets should be limited to around 160 words (probably less, given that code contains unusual characters for non-programmers). It will be necessary for us to judge snippets for inclusion based on how quickly we believe they can be copied (not too quickly for experts and not too slowly for naïve participants).

A switching delay is imposed each time participants switch from the code screen to the copy screen. This delay plays the same role as the physical separation of chessboards in Chase and Simon's study [2]. By ensuring that the two chessboards could not simultaneously be in view, the authors forced participants to utilize their working memories during the copy task. Following deGroot and Gobet, we will use a delay of 5 seconds for each switch [5].

4.3.2 Expected Results

Naïve and novices programmers have been observed to closely follow the surface-level (textual) features of code [16, 6]. Thus, it is expected that they will encode the stimuli in both conditions (normal and scrambled) *linguistically* instead of visually. Given this, we should see activation in the left frontal and parietal regions as well as the bilateral caudate nucleus [18]. Experts, on the other hand, are known to possess chunks for common code patterns and to use these chunks during recall tasks [20]. If their chunks include syntactic and indentation information (i.e., are visual in nature), then we should expect more activation for experts in brain areas associated with visual working memory. Specifically, we expect to see more activation in ventrolateral and dorsal prefrontal cortex (relative to the less experienced programmers) in the normal condition [22]. Given Gobet's hypothesis that expertise largely involves the accumulation of more chunks [10], we hypothesize

that the amount of activation in visual working memory areas will roughly follow programming expertise (again, in the normal condition only). It is unknown whether the increase in activation for visual working memory in experts will also mean a *decrease* in activation for linguistic areas relative to novices.

In a review of the behavioral program comprehension literature, Gobet and Oliver (2002) noted that the expertise effect in program recall (where participants must recall a program verbatim) decreases or virtually disappears when the code is scrambled [10]. The extent to which the effect decreases depends on whether the scrambling is done with modules or individual lines (or both). We expect that scrambling the code by individual lines will force experts to process the stimuli in a manner similar to novices and naïve participants (i.e. linguistically). Thus, we hypothesize that activation in linguistic and visual working memory areas will be similar for naïve/novice participants in both conditions and similar between experts and naïve/novice participants in the scrambled condition.

4.4 Other Methodological Considerations

4.4.1 Code Indentation and Highlighting

The indentation and highlighting of the presented code are important to consider, since they are almost always present when editing code. If chunks are represented visually, then these features could be critical for proper retrieval of the correct chunk (Parnin suggests that both indentation and syntax highlighting help produce a unique signature [17]). One study has found that syntax highlighting does not affect overall visual search performance, but it can enhance searches for particular kinds of targets [11]. To avoid confounding effects from any one highlighting scheme, our code will be presented as simple black text on a white background with only language keywords highlighted.

Indentation will be retained for both normal and scrambled code, since it is often either mandated by the language (e.g., Python), or automatically inserted by the programming environment. For some languages, however, indentation is optional (e.g., Java, C++). This poses a potential problem for our experiment, since individual participants may use very different indentation styles in their own code (and thus have very different chunks). If this appears to be a problem in our pilot study, we plan to reformat all applicable code snippets using a standard indentation style, such as K&R 1TBS [21], to cover as broad a range of participants as possible.

4.4.2 Lateralization

We do not rule out the possibility that hemispherical lateralization may play a significant role in our results. While the left hemisphere has been traditionally associated with linguistic processing, the right hemisphere appears crucial for "holistic" understanding of complex linguistic phenomena [24]. Thus, instead of just finding a simple visual versus linguistic effect for programming expertise, we may also find a strong lateralization effect between experts and novices in linguistic regions (because experts are better at seeing the "bigger picture" behind the code).

4.4.3 Estimating Chunk Capacity

We do not attempt to estimate the capacity of short-term memory in this study. It would be possible to do so with the data we plan to gather, provided an operational definition of a code chunk is agreed upon. In Chase and Simon's chessboard copy task, they defined a chunk using the latency between piece placements. For the code copy task, it is not clear if the latency between key presses could play the same role. Because all keystrokes will be recorded and timestamped, it will be possible to perform a post-hoc analysis to estimate short-term memory capacity. Given the large differences in capacity estimates between Chase et al. and deGroot et al. for the chessboard copy task, it might be useful to have an estimate from a different task.

4.4.4 Verbatim Copying

The requirement that code be copied verbatim may be too strict, and cause some participants to get frustrated (especially those with little programming experience). It may be necessary to include some notion of "close enough" in the computer process that checks for copy correctness, such as ignoring small misspellings in variable names. We will investigate such matters further if it becomes necessary after a small pilot study.

References

[1] W. Barfield. Expert-novice differences for software: Implications for problem-solving and knowledge acquisition. *Behaviour & Information Technology*, 5(1):15–29, 1986.

[2] William G. Chase and Herbert A. Simon. Perception in chess. *Cognitive Psychology*, 4(1):55 – 81, 1973.

[3] Google Code. `http://code.google.com`, Nov 2011.

[4] Bill Curtis. Fifteen years of psychology in software engineering: Individual differences and cognitive science. *Proceedings of the 7th international conference on Software engineering*, pages 97–106, 03 1984.

[5] A. de Groot, F. Gobet, and R. Jongman. *Perception and memory in chess: Studies in the heuristics of the professional eye*. Van Gorcum & Co, Assen, Netherlands, 1996.

[6] Françoise Détienne. *Software design—cognitive aspects*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[7] Françoise Détienne. What model(s) for program understanding? In *Conference on Using Complex Information Systems - UCIS'96*, Poitiers, France, September 1996.

[8] Franoise Dtienne and Elliot Soloway. An empirically-derived control structure for the process of program understanding. *International Journal of Man-Machine Studies*, 33(3):323 – 342, 1990.

[9] F. Gobet and G. Clarkson. Chunks in expert memory: Evidence for the magical number four or is it two? *Memory*, 12(6):732–747, 2004.

[10] F. Gobet and I. Oliver. A simulation of memory for computer programs. *Department of Psychology, ESRC Centre for Research in Development, Instruction and Training, University of Nottingham (UK), Technical report*, 74, 2002.

[11] T. Hakala, P. Nykyri, and J. Sajaniemi. An experiment on the effects of program code highlighting on visual search for local patterns. 2008.

[12] M.H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Ltd, 1977.

[13] C.M. Karat, C. Halverson, D. Horn, and J. Karat. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 568–575. ACM, 1999.

[14] J. Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[15] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[16] J.F. Pane and B.A. Myers. The influence of the psychology of programming on a language design: Project status report. 2006.

[17] Chris Parnin. A cognitive neuroscience perspective on memory for programming tasks. In *In In the Proceedings of the 22nd Annual Meeting of the Psychology of Programming Interest Group (PPIG)*, 2010.

[18] Diana Rodriguez-Moreno and Joy Hirsch. The dynamics of deductive reasoning: An fmri investigation. *Neuropsychologia*, 47(4):949 – 961, 2009.

[19] B. Shneiderman and R. Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Parallel Programming*, 8(3):219–238, 1979.

[20] E. Soloway and K. Ehrlich. Empirical studies of programming knowledge. *IEEE Trans. Software Eng.*, 10, 1984.

[21] Indent Style. `http://en.wikipedia.org/wiki/Indent_style#Variant:_1TBS`, Nov 2011.

[22] L.G. Ungerleider, S.M. Courtney, and J.V. Haxby. A neural system for human visual working memory. *Proceedings of the National Academy of Sciences*, 95(3):883, 1998.

[23] Teun A. van Dijk and W. Kintsch. *Strategies of discourse comprehension*. New York: Academic Press, 1983.

[24] W. Wapner, S. Hamby, and H. Gardner. The role of the right hemisphere in the apprehension of complex linguistic materials. *Brain and Language*, 14(1):15–33, 1981.