Preprint accepted at ICSE 2014.

Understanding Understanding Source Code with Functional Magnetic Resonance Imaging

Janet Siegmund[π,*] Christian Kästner[ω], Sven Apel[π], Chris Parnin[β], Anja Bethmann[θ], Thomas Leich[δ], Gunter Saake[σ], and André Brechmann[θ]
[π]University of Passau, Germany [ω]Carnegie Mellon University, USA
[β]Georgia Institute of Technology, USA [θ]Leibniz Inst. for Neurobiology Magdeburg, Germany
[δ]Metop Research Institute, Magdeburg, Germany [σ]University of Magdeburg, Germany

ABSTRACT

Program comprehension is an important cognitive process that inherently eludes direct measurement. Thus, researchers are struggling with providing suitable programming languages, tools, or coding conventions to support developers in their everyday work. In this paper, we explore whether *functional magnetic resonance imaging (fMRI)*, which is well established in cognitive neuroscience, is feasible to more directly measure program comprehension. In a controlled experiment, we observed 17 participants inside an fMRI scanner while they were comprehending short source-code snippets, which we contrasted with locating syntax errors. We found a clear, distinct activation pattern of five brain regions, which are related to working memory, attention, and language processing—all processes that fit well to our understanding of program comprehension. Our results encourage us and, hopefully, other researchers to use fMRI in future studies to measure program comprehension and, in the long run, answer questions, such as: Can we predict whether someone will be an excellent programmer? How effective are new languages and tools for program understanding? How should we train developers?

1. INTRODUCTION

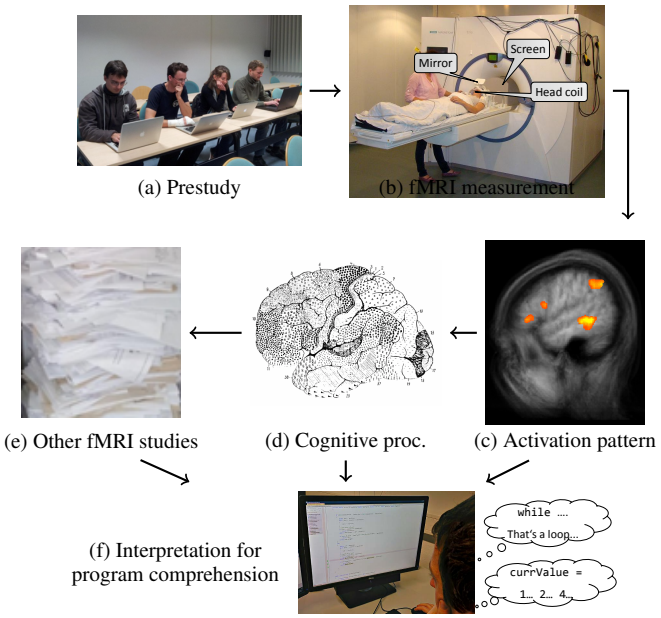
As the world becomes increasingly dependent on the billions lines of code written by software developers, little comfort can be taken in the fact that we still have no fundamental understanding of how developers understand source code.

Understanding program comprehension is not limited to theory building, but can have real downstream effects in improving education, training, and the design and evaluation of tools and languages for programmers. If direct measures of cognitive effort and difficulty could be obtained and correlated with programming activity, then researchers could identify and quantify which types of activities, segments of code, or kinds of problem solving are troublesome or improved with the introduction of a new language or tool.

In studying programmers, decades of psychological and observational experiments have relied on indirect techniques, such as comparing task performance or having programmers articulate their thoughts in think-aloud protocols. Each method, when skillfully applied, can yield important insights. However, these common techniques are not without problems. In human studies of programming, individual [13] and task variance [18] in performance often mask any significant effects hoping to be found when evaluating, say, a new tool. Think-aloud protocols and surveys rely on self-reporting and require considerable manual transcription and analysis that garner valuable but indefinite and inconsistent insight.

In the past few decades, psychologists and cognitive neuroscientists have collectively embraced methods that measure physiological correlates of cognition as a standard practice. One such method is *functional magnetic resonance imaging (fMRI)*, a non-invasive means of measuring blood-oxygenation levels that change as a result of localized brain activity.

In this paper, we report on results and experience from applying fMRI in a program-comprehension experiment. While our experiment is a first step toward measuring program comprehension with fMRI, and as such inherently limited, we believe this study can illuminate a path toward future studies that systematically explore hypotheses and that can be used to build stronger theories of program comprehension.



(a) Prestudy (b) fMRI measurement (c) Activation pattern (d) Cognitive proc. (e) Other fMRI studies (f) Interpretation for program comprehension

Figure 1: Workflow of our fMRI study.

*This author published previous work as Janet Feigenspan.

In our experiment, 17 participants performed two kinds of tasks while in an fMRI scanner. In the first kind (referred to as *comprehension tasks*), developers should comprehend code snippets and identify the program's output. In the second kind (referred to as *syntax tasks*), developers identified syntax errors in code snippets, which is similar to the comprehension tasks, but does not require actual understanding of the program. As a result of our study, we found:

- evidence that distinct cognitive processes were used when performing the comprehension tasks,
- activation of functional areas related to working memory, attention, and language comprehension, and
- a left-hemisphere lateralization.

Our results provide direct evidence of the involvement of working memory and language processing in program comprehension, and suggest that, while learning programming, training working memory (necessary for many cognitive tasks), and language skills (which Dijkstra already claimed as relevant for programming), might also be beneficial for programming skills. Furthermore, our results can help to validate or invalidate particular theories of program comprehension. Although a single study is not sufficient to answer general questions, we can raise some further probing questions: If program comprehension is linked to language comprehension, does learning and understanding a programming language require the same struggles and challenges as learning another spoken language? If program comprehension only activates the left hemisphere (often referred to as analytical), can we derive guidelines on how to train students?

Taking a broader perspective, our study demonstrates the feasibility of using fMRI experiments in software engineering. Although we believe this is only a first step, our experience and experimental design is meant to be a template for other researchers to adopt and improve. With decreasing costs of fMRI studies, we believe that such studies will become a standard tool also in software-engineering research.

There are still many interesting, unanswered questions that can follow this line of research: How do people use domain knowledge during comprehension? To what extent is implementing source code a creative process? Can we train anybody to become an excellent programmer? How should we design programming languages and tools for optimal developer support? Can software metrics predict the comprehensibility of source code?

In summary, we make the following contributions:

- We designed the first fMRI study to observe brain activity during comprehension tasks. We share our design and experiences.
- We conducted the study with 17 participants and observed activation in five distinct brain regions. This demonstrates the potential of fMRI studies in software-engineering research.
- We interpret how the identified cognitive processes contribute to program comprehension and discuss future research directions.

2. FMRI STUDIES IN A NUTSHELL

Rationale of fMRI Studies. The first goal in fMRI studies is to locate and isolate brain activity associated with a behavior. To identify a region of the brain, scientists use instruments with high spatial precision, such as fMRI scanners. After having established a general idea of where brain activity is occurring, scientists further try identifying the timing and interaction of brain activity between different brain locations. For example, scientists will try to measure the time to process a color or a word.

Complex behaviors, such as understanding a spoken sentence, require interactions among multiple areas of the brain. Eventually, to create a model of behavior, scientists use techniques to dissociate activated brain areas to understand how a particular brain area contributes to a behavior. For instance, scientists found that the *left medial extra striate cortex* was associated with visual processing of words and pseudo words that obey English spelling, but not activated by unfamiliar strings of letters or letter-like forms [54].

To reference an identified brain location, Brodmann areas have proven useful as classification system. There are 52[1] Brodmann areas [11], each associated with cognitive processes, such as seeing words or retrieving meaning from memory. Through extensive research in this field over the past twenty years, there is a detailed and continuously growing map between Brodmann areas and associated cognitive processes (e.g., `www.cognitiveatlas.org` shows an atlas). Due to its success, we selected fMRI to evaluate whether it is feasible to measure program comprehension.

Now, given such map, if we study a new task, such as program comprehension, we can identify which brain regions are activated and consequently hypothesize which cognitive processes are involved. For example, we found that one of the activated regions in our study is related to language recognition, so we can hypothesize that language recognition is an integral part of program comprehension, which was not certain a priori (see Section 5.2).

General Challenges of fMRI Studies. Studies using fMRI face general challenges due to the technologies involved, which are very different from, say, controlled experiments in empirical software engineering.

fMRI is based on measuring differences in oxygen levels of blood flow in the brain. If a brain region becomes active, its oxygen need increases, and the amount of oxygenated blood in that region increases, while the amount of deoxygenated blood decreases—which is known as the *BOLD (blood oxygenation level dependent)* effect. Oxygenated and deoxygenated blood have different magnetic properties, which are measured by fMRI to identify activated brain regions.

The BOLD effect needs a few seconds to manifest. Typically, after about 5 seconds, it peaks; after a task is finished, the oxygen level returns to the baseline level after 12 seconds. Often, before returning to the baseline, the oxygen level drops below the baseline [36]. Thus, the length of the experiment has to be planned carefully. For optimal measurement of the BOLD effect, task durations between 30 and 120 seconds have proven useful, followed by a rest condition of about 30 to 60 seconds. Longer task duration allows the BOLD signal to accumulate, which produces better differences between tasks. Furthermore, we need several measurements, so an experiment consists of several similar tasks to average the BOLD effect over all tasks. This way, we can make statistically sound claims about the BOLD effect.

To unambiguously determine the brain region in which the BOLD effect took place, we need to avoid motion artifacts, that is, noise that occurs when participants move their head. To this end, participants are instructed to lie as motionless as possible during the measurement, and the head is fixed with cushions. Furthermore, communication and interaction with participants is limited, because speaking or pressing buttons leads to further motion artifacts. In such a restricted setting, the experiment duration should

[1] Some areas can divided further, e.g., 23a and 23b.

```
1 public static void main(String[] args) {
2   String word = "Hello";
3   String result = new String();
4
5   for (int j = word.length() - 1; j >= 0; j--)
6     result = result + word.charAt(j);
7
8   System.out.println(result);
9 }
```

Figure 2: Source code for one comprehension task with expected output ‘olleH‘.

```
1 public static void main(String[] ) {
2   String word = "Hello';
3   String result = new String();
4
5   for (int j = word.length() - 1; j >= 0; j--)
6     result = result + word.charAt(j);
7
8   System.out.println{result);
9 }
```

Figure 3: Source code for syntax task with errors in Line 1, 2, and 8.

not exceed one hour, because after that, participants start getting restless.

Additionally, participants can see a relatively small screen reflected through a mirror (illustrated in Figure 1b), which cannot reasonably show more than 20 lines of text.

Finally, we need to distinguish brain activations caused by the experimental tasks from other activations. In tasks that require participants to watch a screen or listen to a signal, there will be activations caused by visual or audio processing. To filter activations that are not specific for the experimental tasks, we need to design control tasks that are as similar as possible to the experimental tasks and differ only in the absence of the targeted cognitive process.

Requirements for Our fMRI Study. For a study on program comprehension, the general fMRI challenges translate into a specific set of requirements.

First, due to the small mirror in the fMRI scanner, we can show only a limited amount of source code at a time. Technically, it is possible to let participants scroll, but that would cause motion artifacts.

Second, we need source-code fragments with a suitable difficulty. If source code is too easy to understand, then participants may finish too early, such that the BOLD activation returns to the baseline before the end of a trial. On the other hand, if source code is too difficult, participants cannot finish understanding it. In this case, we cannot be sure that the cognitive process actually took place long enough to be measured. The challenge is to find the right level of difficulty—short code fragments that require 30 to 120 seconds to understand. So, in a one-hour experiment, we can perform about a dozen repetitions, for which we need comparable tasks of similar difficulty.

Finally, to filter out irrelevant activation, we need control tasks that ideally differ from the comprehension task only in the absence of comprehension, nothing else. In our context, control tasks are different from typical control tasks in software-engineering experiments, where a baseline tool or language is used; in fMRI, the similarity is defined on a low, fine-grained level, such that we can observe the activation caused by comprehension only.

These constraints—short code fragments of controlled difficulty and limited repetitions—impair external validity, as we discuss in Section 7. Results of fMRI studies can be generalized to realistic situations only with care.

Our fMRI Study. Given the constraints, we selected short algorithms that are taught in first-year undergraduate computer-science courses as *comprehension tasks*, such as the string-reversal code in Figure 2. We asked participants to determine the output of the program ("olleH", in our example), which they can accomplish only if they understand the source code. The programs we used included sorting and searching in arrays, string operations, and simple integer arithmetic. To account for different domain knowledge of participants, we excluded its influence by obfuscating identifiers and enforcing program comprehension that required understanding code from the bottom-up, that is, from syntax to semantics (see Sec 3.1).

As control tasks (*syntax tasks*), we introduced syntax errors, such as quotation marks or parentheses that do not match and missing semicolons or identifiers, into the same code fragments as for the comprehension tasks (illustrated in Figure 3). Then, we asked participants to find syntax errors (Lines 1, 2, and 8). Comprehension and syntax tasks are very similar, yet sufficiently different: Both require the participants to look at almost identical pieces of text, but for the syntax tasks, participants do not need to *understand* the code.

To find suitable comprehension and syntax tasks, we conducted pilot studies in a computer lab (Figure 1a). We let a total of 50 participants solve 23 comprehension tasks and search for more than 50 syntax errors. Based on our observations, we selected 12 source-code snippets and corresponding syntax errors with suitable duration and difficulty.

For the actual study (Figure 1b), we conducted the experiment with 17 participants inside an fMRI scanner. Although initial fMRI studies often do not yield conclusive results because of missing empirical evidence, such as related studies, and hypotheses about involved areas, to our surprise, we measured a clear activation pattern (Figure 1c), which is a very encouraging result that we discuss in Section 5.

3. STUDY DESIGN

Having provided a high-level overview, we now present the technical details of our study. Additional material (e.g., all source-code snippets) is available at the project's website.[2] Readers interested only in the big picture and the results may skip Section 3.

3.1 Objective

To the best of our knowledge, we performed the first fMRI study to measure program comprehension. Since we are exploring our options, we do not state specific research hypotheses about activated brain regions, but, instead, pose a research question:

RQ: What brain regions are activated during program comprehension?

To soundly evaluate this question, we first need to take a look at the complexity of the comprehension process. There are, roughly spoken, two classes of comprehension models: top-down comprehension [12, 70] and bottom-up comprehension [53, 65]. Top-down comprehension describes that, when programmers are familiar with a program's domain, they use their domain knowledge to understand source code. During that process, *beacons* (e.g., familiar

[2] tinyurl.com/ProgramComprehensionAndfMRI/

identifier names) help to form hypotheses about a program's purpose. If developers cannot apply their domain knowledge, they use bottom-up comprehension, so they understand source code statement by statement. Since differences in domain knowledge are hard to control, we expect more noise in top-down comprehension. To avoid any possible additional activation, we focus on bottom-up comprehension.

3.2 Experimental Design

All participants completed the experiment in the same order. Before the measurement, we explained the procedure to each participant and they signed an informed consent form. Each session started with an anatomical measurement stage that lasted 9 minutes. This was necessary to map the observed activation to the correct brain regions.

Next, participants solved tasks inside the fMRI scanner in the Leibniz Institute for Neurobiology in Magdeburg. We had 12 trials, each consisting of a comprehension task and a syntax task, separated by rest periods:

1. Comprehension task [60 seconds]
2. Rest [30 seconds]
3. Syntax task [30 seconds]
4. Rest [30 seconds]

The rest periods, in which participants were instructed to do nothing, was our baseline (i.e., the activation pattern when no specific cognitive processes take place). To familiarize participants with the setting, we started with a warming-up trial, a hello-world example that was not analyzed.

Instead of saying or entering the output of source-code snippets, participants indicated when they have determined the output in their mind or located all syntax errors by using the left of two keys of a response box with their right index finger. Directly after the scanning session, participants saw the source code again on a laptop and entered their answer to ensure that comprehension took place. With this procedure, we minimized motion artifacts inside the fMRI scanner.

3.3 Material

Initially, we selected 23 typical algorithms that are typically taught in first-year undergraduate computer-science education at German universities. For example, we had algorithms sorting or searching in arrays, string operations (cf. Fig. 2), and simple integer arithmetic, such as computing a power function and determining whether a number is prime (see project's website for all initially selected source-code snippets). The selected algorithms were different enough to avoid learning effects from one algorithm to another, but yet similar enough (e.g., regarding length, difficulty) to elicit similar activation, which is necessary for averaging the BOLD effects for all tasks.

We created a main program for each algorithm, printing the output for a sample input. All algorithms are written in imperative Java code inside a single main function without recursion and with light usage of standard API functions. To minimize cognitive load caused by complex operations that are not inherent to program comprehension, we used small inputs and simple arithmetic (e.g., 2 to the power of 3). To avoid influences due to domain knowledge and possible brain activation caused by memory retrieval, we obfuscated identifier names, so that participants needed bottom-up comprehension to understand the source code. For example, in Figure 2, the variable `result` does not give a hint about its content (i.e., that it holds the reversed word), but only about its purpose (i.e., that it contains the result).

We injected three syntax errors into every program to derive control tasks that are otherwise identical to the corresponding comprehension tasks, as illustrated in Figure 3. The syntax errors we introduced can be located without understanding the execution of the program, they merely require some kind of pattern matching.

In a first pilot study [67], we determined whether the tasks have suitable difficulty and length. In a lab session, we asked participants to enter the output of the source-code snippets and measured time and correctness. 41 undergraduate computer-science students of the University of Passau participated. To simulate the situation in the fMRI scanner, participants were not allowed to make any notes during comprehension. Based on the response time of the participants, we excluded six snippets with a too high mean response time (>120 seconds) and one snippet with a too low response time (<30 seconds). We also analyzed correctness, to ensure that the snippets are not too difficult and that comprehension actually took place. On average, 90 % of the participants correctly determined the output, so we did not exclude any snippets based on difficulty.

In a second pilot study, we evaluated the suitability of syntax tasks, so that we can isolate the activation caused only by comprehension. Undergraduate students from the University of Marburg (4) and Magdeburg (4), as well as one professional Java programmer located syntax errors. We analyzed response time and correctness to select suitable syntax tasks. All response times were within the necessary range, and most participants found, at least, two syntax errors. Thus, the syntax tasks had a suitable level of difficulty.

For the session in the scanner, we further excluded four tasks to keep the experiment time within 1 hour. We excluded one task with the shortest and one with the longest response time. We also excluded two tasks that are similar to other tasks (e.g., adding vs. multiplying numbers).

We presented the source-code snippets in a fixed order. Whenever possible, we let participants first comprehend a snippet, then, in a later trial, locate syntax errors in the corresponding snippets, with a large as possible distance between both (see project's website for complete ordering). This way, we minimized learning effects.

Furthermore, we assessed the programming experience of participants with an empirically developed questionnaire to assure a homogeneous level of programming experience [23], and we assessed the handedness of our participants with the Edinburgh Handedness Inventory [49], because the handedness correlates with the role of the brain hemispheres [42] and, thus, is necessary to correctly analyze the activation patterns.

3.4 Participants

To recruit participants, we used message boards of the University of Magdeburg. We recruited 17 computer-science and mathematics students, two of them were female, all with an undergraduate level of programming experience and undergraduate level of Java experience (see projects website for details), comparable to our pilot-study participants. Thus, we can assume that our participants were able to understand the algorithms within the given time frame. We selected students, because they are rather homogeneous; this way, the influence of different backgrounds is minimized.

All participants had normal or corrected-to-normal vision. One participant was left handed, but showed the same lateralization as right handers, as we determined by a standard lateralization test [7]. Participants gave written informed consent to the study, which was approved by the ethics committee of the University of Magdeburg. As compensation, participants were paid 20 Euros. Finally, participants were aware that they could end the experiment at any time.

3.5 Imaging Methods

The imaging methods describe the standard procedure of fMRI studies.

Source-Code Presentation. For source-code presentation and participant-response recording, we used the Presentation software (`www.neurobs.com`) running on a standard PC. Source code was back-projected onto a screen that could be viewed via a mirror mounted on the head coil (cf. Fig. 1b). The distance between the participant's eyes and the screen was 59 cm, with a screen size of 325×260 mm, which is appropriate for an angle of $\pm 15^\circ$. The source-code snippets were presented in the center of the screen with a font size of 18, as defined in the Presentation software. The longest source-code snippet had 18 lines of code.

Data Acquisition. We carried out the measurements on a 3 Tesla scanner (Siemens Trio, Erlangen, Germany) equipped with an eight channel head coil. The 3D anatomical data set of the participant's brain (192 slices of 1 mm each) was obtained before the fMRI measurement. Additionally, we acquired an Inversion-Recovery-Echo-Planar-Imaging (IR-EPI) scan with the identical geometry as in the fMRI measurement, to obtain a more precise alignment of the functional to the 3D anatomical data set.

For fMRI, we acquired 985 functional volumes in 32 minutes and 50 seconds using an echo planar imaging (EPI) sequence (echo time (TE), 30 ms; repetition time (TR), 2000 ms; flip angel, $\pm 80^\circ$; matrix size, 64×64; field of view, $19.2\,\text{cm} \times 19.2\,\text{cm}$; 33 slices of 3 mm thickness with 0.45 mm gaps). During the scans, participants wore earplugs for noise protection and their head was fixed with a cushion.

Data Preparation. We analyzed the functional data with BrainVoyager™QX 2.1.2 (`www.brainvoyager.com`). We started a standard sequence of preprocessing steps, including 3D-motion correction (where each functional volume is coregistered to the first volume of the series), linear trend removal, and filtering with a high pass of three cycles per scan. This way, we reduced the influence of artifacts that are unavoidable in fMRI studies (e.g., minimal movement of participants). Furthermore, we transformed the anatomical data of each participant to a standard Talairach brain [72]. This way, we can average the BOLD effect over all participants (see next paragraph).

Analysis Procedure. We projected the functional data set to the IR-EPI images and co-registered these with the 3D-data set. Then, we transformed the fMRI data to Talairach space and spatially smoothed them with a Gaussian filter (FWHM=4 mm). For the random-effects GLM analysis, we defined one predictor for the comprehension tasks and one for the syntax tasks. These were convolved with the two-gamma hemodynamic response function using the default parameters implemented in BrainVoyager™QX. We averaged the hemodynamic response for each condition (comprehension, syntax) across the repetitions. Furthermore, we normalized the BOLD response to the baseline that is defined by averaging the BOLD amplitude 15 seconds before the onset of the comprehension and syntax condition, respectively. Then, we averaged the BOLD response over all participants.

Next, we contrasted comprehension with the rest condition using a significance level of $p < 0.05$ (FDR-corrected [5]), to determine the voxels that indeed showed a positive deflection of the BOLD response, compared to the rest period (a negative deflection does not show a real activation, so only the positive deflections are of interest). These voxels comprised a mask, which was used in the subsequent contrast, where we directly compared comprehension with syntax tasks at a significance level of $p < 0.01$ (FDR-corrected) and a minimum cluster size of $64\,\text{mm}^3$.

As the last step, we determined the Brodmann areas based on the Talairach coordinates with the Talairach daemon (client version, available online at `www.talairach.org`). The Talairach space is used for the technical details of the analysis, and the Brodmann areas are used to map activated areas to cognitive processes.

4. RESULTS

In Figure 4, we show the resulting activation pattern of the analysis, including the time course of the BOLD responses for each cluster. The activation picture and BOLD responses are averaged over all tasks per condition (comprehension, syntax) and participants; the gray area around the time courses shows the standard deviation based on the participants' averaging.

We included the data of all participants, since all showed comprehension of the source-code snippets by one of three ways: entering the correct output of the source code after the experiment, correctly describing what the source code was doing, or by ensuring that they attempted to comprehend the source code (based on the questionnaire after the measurement; see project's website for details).

In essence, we found five relevant activation clusters, all in the left hemisphere. For each cluster, we show Talairach coordinates, the size of the cluster, related Brodmann areas, and relevant associated cognitive processes (note that deciding which cognitive processes are relevant belongs to the interpretation, not results; see Section 5). Thus, we can answer our research question:

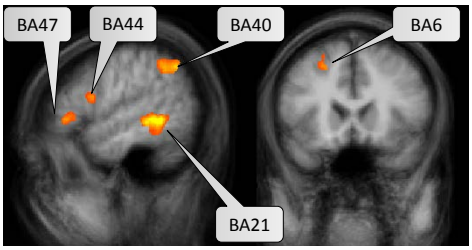
RQ: During program comprehension, Brodmann areas 6, 21, 40, 44, and 47 are activated.

5. DISCUSSION

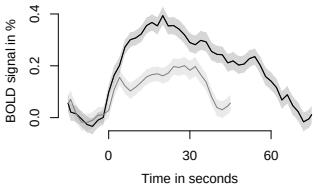
Having presented our results, we interpret the activation pattern. As is standard in fMRI studies, we start with relating relevant cognitive processes to Brodmann areas (Figure 1d). Next, we look at tasks of other fMRI studies, in which similar Brodmann areas were identified, and we relate our findings to previous findings (Figure 1e). Last, we abstract from Brodmann areas and hypothesize how the cognitive processes involved contribute to program comprehension. We conclude with a discussion about what cognitive processes are relevant for program comprehension, based on our and other findings of fMRI studies (Figure 1f).

In our study, we observed activation in five Brodmann areas; overall, there are 52 Brodmann areas that are associated with cognitive processes (cf. Section 2). Thus, finding five activated clusters, which are related to activities that fit well to our understanding of the comprehension process, especially in an initial study, is a surprisingly good result of our study. However, individual Brodmann areas are often associated with multiple cognitive processes. Thus, as part of the interpretation, we discussed among the author team (which included a psychologist, a neurobiologist, a linguist, as well as computer scientists and software engineers) whether a process is relevant or irrelevant for program comprehension. For example, Brodmann area 21 is typically activated when humans give a spoken response. However, our setting did not include a spoken response, so this process and according studies were not relevant for us. Still, for completeness and replication, we mention all associated activities.

Note that, since we only look at the difference of activation pattern between the comprehension and syntax tasks, we only consider

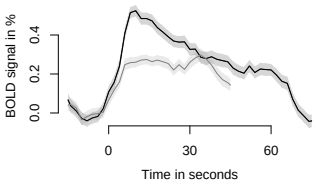


BA 6: Middle frontal gyrus
(Talairach coord.: -26, 17, 52; cluster size: 1279)



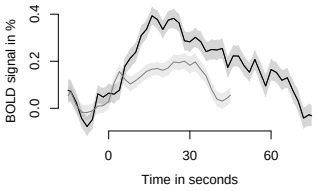
Attention
Division of attention
Language
Silent word reading
Working memory
Verbal/numeric
Problem solving

BA 21: Middle temporal gyrus
(Talairach coord.: -55, -39, -2; cluster size: 4746)



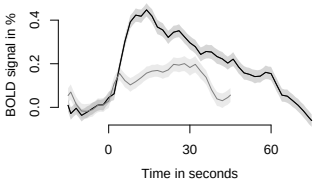
Semantic memory retrieval
Categorization

BA 40: Inferior parietal lobule
(Talairach coord.: -51, -49, 41; cluster size: 3368)



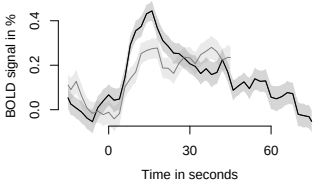
Working memory
Verbal/numeric
Problem solving

BA 44: Inferior frontal gyrus
(Talairach coord.: -50, 11, 16; cluster size: 698)



Working memory
Verbal/numeric

BA 47: Inferior frontal gyrus
(Talairach coord.: -52, 31, 0; cluster size: 546)



Language
Silent word reading
Working memory
Problem solving

Figure 4: Observed activation pattern for program comprehension and time courses of the BOLD response for each cluster. The gray area around the time courses depicts the standard deviation based on the participants. BA: Brodmann area.

activation caused by comprehension. For example, we do not expect (and did not find) activations regarding the visual perception of source code, which is almost identical for both tasks.

5.1 Brodmann Areas 6 and 40

With Brodmann areas (BA) 6 and 40, similar cognitive processes are associated, so we discuss them together. In the context of the current study, the processes *division of attention*, *silent word reading*, *working memory for verbal and/or numerical material*, and *problem solving* are particularly relevant.[3] When discussing each relevant activity, we also describe results of related neuro-imaging studies and the relationship to program comprehension.

Division of attention. Division of attention describes that attention has to be distributed to more than one cognitive activity. Other studies of divided attention also found both, BA 6 and BA 40, activated. For example, Vandenberghe and others let participants discriminate visually presented objects regarding two features (orientation and location) [75]. These activities are similar to what participants did in our study, that is, dealing at the same time with understanding statements of the source code and the numbers or words that the source code was manipulating. In the syntax tasks, participants were only looking for syntax errors, and did not have to divide their attention to hold any values or words.

Silent word reading. In our study, participants read words and, in the comprehension task, needed to understand their meaning (e.g., `for` denotes a loop). Other studies also found both areas activated when participants understood the meaning of sentences, compared to deciding whether a sentence or word is grammatically correct [10, 59]. This is in line with understanding the meaning of source-code statements, compared to analyzing syntactical correctness of statements in our syntax tasks.

Verbal/Numerical Working Memory. Working memory is relevant for many cognitive tasks. One part of working memory, the phonological loop, allows us to memorize spoken information, such as telephone numbers, as long as we repeat it (either spoken aloud or silently) [2]. Numerous studies that let participants use verbal rehearsal found the same Brodmann areas (6 and 40). For example, Smith and others let participants keep letters in mind [69]; Awh and others additionally let participants compare target letters with letters that have been shown two letters earlier in a sequence of letters [1]. In our study, participants had to keep the values of variables that the source code was manipulating in mind to understand the source code. Additionally, when loops were part of the source code, participants had to keep the values of several loop iterations in mind to fully understand what was going on. Both activities were not necessary when identifying syntax errors. Thus, the phonological loop fits well to our understanding of bottom-up program comprehension.

Problem Solving. Problem solving is a broad term that captures several similar tasks, for example, the Wisconsin Card Sorting test [6] or Raven's Progressive Matrices [60]. Both tests require participants to abstract from presented patterns and discover the

[3]Other associated activities that do not appear relevant are (a) attention regarding orientation and stimulus-response compatibility, (b) space/motion perception and imagery, (c) spatial and object-related working memory, (d) episodic memory encoding of objects and space, (e) episodic memory retrieval of context information as well as retrieval effort, and (f) skill learning of unpracticed (non-)motor skills.

rules that construct the material. This is similar to our comprehension tasks, in which participants need to abstract from statements in source code and discover how and why these statements work together, which is not necessary for locating syntax errors. Other fMRI studies also found BA 6 and 40, for example when using the above-mentioned Wisconsin Card Sorting test [47] or Raven's Progressive Matrices [58].

Overall, BA 6 and 40 fit well into our understanding of program comprehension. By consulting related studies, we found related processes that capture the multiple facets of understanding source code. The remaining three Brodmann areas are often found in language-based experiments, so we discuss them together.

5.2 Brodmann Areas 21, 44, and 47

In addition to other cognitive processes,[4] BA 21, 44, and 47 are related to different facets of language processing. Numerous studies showed the involvement of all three Brodmann areas in artificial as well as natural-language processing [3, 55, 68]. In particular, artificial-language processing is interesting, because artificial languages are based on formal grammars and limited sets of symbols, such as words or graphemes, from which letter or word sequences are created. Participants of typical artificial-language studies should decide based on their intuition, after a learning period, whether sequences are grammatical or not, resulting in activation in BA 21, 44, and 47. Artificial-language processing and program comprehension are similar, since both usually built on a limited set of elements and rules; in the syntax tasks, participants had to do some kind of pattern matching to locate the syntax errors. Based on the similarity of program comprehension to artificial-language processing, which is in turn similar to natural-language processing, we conjecture that one part of program comprehension likely involves language processing.

The *posterior middle temporal gyrus (MTG)* (BA 21) is closely associated with semantic processing at the word level. Both imaging and lesion studies suggest an intimate relation between the success or failure to access semantic information and the posterior MTG [9, 21, 74]. In our study, participants also needed to identify the meaning of written words in the source code to successfully understand the source code and its output, which was not necessary for the syntax tasks. Thus, we found evidence that understanding the meaning of single words is a necessary part of program comprehension. This may not sound too surprising, but we *actually observed* it in a controlled setting.

The *inferior frontal gyrus (IFG)* (BA 44 and 47) is related to combinatorial aspects in language processing, for example, processing of complex grammatical dependencies in sentences during syntactic processing [24, 28]. Several studies suggest that real-time combinatorial operations in the IFG incorporate the current state of processing and incoming information into a new state of processing [30, 56]. Hence, the IFG was proposed to be involved in the unification of individual semantic features into an overall representation at the multi-word level [74]. This is closely related to bottom-up program comprehension, where participants combine words and statements to semantic chunks to understand what the source code is doing. In the syntax tasks, participants did not need to group anything to succeed.

In addition to the individual Brodmann areas, there is evidence for a direct interaction between the activated areas of our comprehension task. Two separate clusters were activated in the IFG, one in BA 44 and one in BA 47, which is also suggested by other fMRI studies. BA 44 was mainly associated with core syntactic processes, such as syntactic structure building [24, 25, 26]. In contrast, BA 47 is assumed to serve as a semantic executive system that regulates and controls retrieval, selection, and evaluation of semantic information [62, 74]. Accordingly, program comprehension requires the participants to build up the underlying syntactic structures, to retrieve the meanings of the words and symbols, and to compare and evaluate possible alternatives; none of these processes is necessary to locate syntax errors.

Moreover, reciprocal connections via a strong fiber pathway between BA 47 and the posterior MTG—the *inferior occipito-frontal fasciculus*—have been claimed to support the interaction between these areas, such that appropriate lexical-semantic representation are selected, sustained in short-term memory throughout sentence processing, and integrated into the overall context [74]. Regarding program comprehension, we conjecture that, to combine words or symbols to statements, and statements to semantic chunks, the neural pathway between the MTG and IFG is involved.

6. IMPLICATIONS FOR PROGRAM COMPREHENSION

Having identified the clusters and related them to other fMRI studies and specific activities of program comprehension, we now combine all findings to a high-level understanding of bottom-up program comprehension.

Working Memory and Divided Attention. First, we found areas related to working memory, especially the phonological loop of verbal/numerical material and problem solving. Regarding the phonological loop, we assume that participants needed to keep the value of numbers or words in mind, while going through the source code statement by statement, dividing their attention during this process. Furthermore, we found a relationship to problem-solving activities. In our experiment, we enforced bottom-up comprehension by obfuscating identifier names, which shows similarities to the Wisconsin Card Sorting Test [6] and Raven's Progressive Matrices [60]. In both tests, as well as during bottom-up comprehension, participants need to understand underlying rules; how cards are sorted, how figures are created, or how loops terminate and when, how and where characters or numbers are modified. Furthermore, participants need to apply these rules, such that they can sort cards, continue rows of figures, or determine the correct output. Thus, our findings align well with the common understanding of bottom-up comprehension, in which rules and relationships in source code have to be discovered and applied.

Consequently, to become excellent in bottom-up comprehension, we might need to train working memory capacity, divided attention, and problem-solving ability.

Language Processing. Second, we found a strong relation to language processing. To understand source code, participants had to process single words and symbols as well as statements that consist of single words (located in the posterior MTG). For example, in Figure 2, Line 6, participants needed to process the words and symbols `result`, `=`, `word`, `+`, and `charAt`, and combine all to understand what this line is doing (i.e., adding a single character

[4] Again, all areas are activated during different cognitive processes. Most likely irrelevant processes are (a) object perception, (b) spoken word recognition, (c) written word recognition with spoken response, (d) object-related working memory, (e) episodic memory encoding of objects, (f) episodic memory retrieval of nonverbal material regarding retrieval mode and effort, (g) conceptual priming, and (h) skill learning of unpracticed motor skills. Again, we only discuss processes relevant to program comprehension.

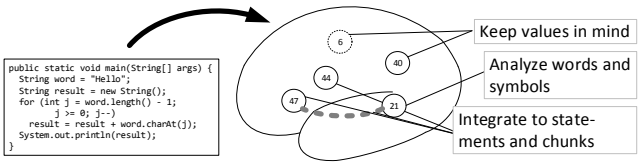


Figure 5: Visualization of how program comprehension might take place.

to the result). Additionally, participants had to integrate all statements to a semantic chunk that reverses a word. Via strong neural pathways, both areas are connected, which is necessary to integrate words/symbols to statements and statements to semantic chunks. Hence, our results support Dijkstra's claim that good programmers need to be exceptionally good in their native tongue [20]. Consequently, focus on language learning in early childhood might facilitate learning programming.

Toward a Model of Bottom-up Comprehension. We can hypothesize what a cognitive model of bottom-up program comprehension can look like: Participants analyze words/symbols of source code, combine symbols to statements, then statements to semantic chunks. Simultaneously to comprehending and integrating words/symbols and statements, participants keep the values of words and numbers in their phonological loop to correctly understand the source code. In Figure 5, we illustrate this process.

Based on this model, we can hypothesize what influences program comprehension. For example, if we increase the number of variables beyond the capacity of the phonological loop, program comprehension should be impaired (more discussion in Section 8).

7. THREATS TO VALIDITY

The challenges and requirements of fMRI studies give rise to several threats to validity. First, we performed several steps to interpret the data. Especially, when deciding which cognitive processes for each Brodmann area are relevant, we might have missed important processes. As a consequence, our interpretation might have led to a different comprehension model. To reduce this threat, we discussed among the author team, which combines the expertise of psychology, neurobiology, linguistics, as well as computer science and software engineering, for each process whether it might be related to our comprehension tasks. Furthermore, we mentioned all processes that are known to be associated with these Brodmann areas (Footnotes 3 and 4), so that readers can make their own judgment about relevant processes.

Second, the source-code snippets that we selected were comparatively short, at most, 18 lines of code. Furthermore, we focused on bottom-up comprehension to avoid any additional activation patterns. Both reflect only one aspect of the complex comprehension process. Thus, we cannot generalize our results to large-scale real-world programming activity—clearly, more studies have to follow. Additionally, we recruited undergraduate students as participants, so our results can be interpreted only in terms of this level of programming experience. However, working with small source-code snippets and having little domain knowledge is typically how beginners start to learn programming. So, for this context, our results give us important insights.

Furthermore, we cannot be entirely certain to what extent we ensured bottom-up comprehension. It is possible that participants recognized some algorithms, as they were taken from typical introductory courses. However, since we obfuscated identifier names and since we did not observe activation in typical memory-related areas, we believe that the participants used bottom-up comprehension most of the time.

Another threat to external validity is that we kept the background of our participants, such as their programming experience and culture, constant to reduce any noise during measurement. Thus, we can generalize our results only carefully. In the next section, we outline, amongst others, how such personal differences might affect program comprehension.

8. FUTURE DIRECTIONS

With our study, we showed that measuring program comprehension with an fMRI scanner is feasible and can result in a plausible activation pattern. But, how does our study contribute to software-engineering research, education, and practice?

Having shown that measuring program comprehension with fMRI is feasible, we encourage other researchers to apply fMRI in their research. With careful planning and by consulting experts, fMRI can be a valuable tool also in software-engineering research. As the costs and inaccessibility of fMRI studies decrease over time, we believe that they will become a standard tool in the software-engineering researcher's toolbox. We hope that other researchers can profit from our experience, learn about requirements and pitfalls of such studies, or even reuse experimental material, so they do not have to start from scratch.

While our study provides only limited direct answers, it raises many interesting and substantial questions for future research. What happens during top-down comprehension? What happens during implementation of source code? How should we train programmers? How should we design programming languages and tools? Can software metrics capture how difficult source code will be to comprehend?

Top-down comprehension. In our experiment, we focused on bottom-up comprehension to minimize additional activation. In future experiments, we shall create more complex source-code snippets in a familiar domain, such that participants use top-down comprehension and their memory to understand source code. For example, we can show the same source-code snippets without obfuscating identifier names, and observe to what extent they serve as beacons for participants. Additionally, we shall select programs that participants are familiar with. In such a setting, we would expect activation of typical memory areas, such as Brodmann areas 44 and 45 in the inferior frontal gyrus or Brodmann area 10 in the anterior prefrontal cortex [14]. In conjunction with an eye tracker, we shall evaluate whether participants fixate on beacons or familiar elements shorter or longer than unfamiliar statements, and how that gazing is related to neural activity. Digging deeper, we may ask at which experience level beginners start using their previous knowledge? To what extent does the knowledge of concepts, such as design patterns, influence activation patterns?

Implementing source code. What happens when people *implement* source code, instead of only understanding it? Writing source code is a form of synthesizing new information, compared to analytical program comprehension. Consequently, we might observe activation of several right-hemispheric regions, such as right BA 44 and 47 for speech production. It would be interesting to study whether and how writing source code is similar to and different from speech production. Initial evidence suggests that developers had high levels of subvocal speech while editing code [51].

Training. There are many discussions about the best way to teach computer science and software engineering [16, 43, 66]. The close relationship to language processing raises the question of whether it is beneficial to learn a programming language at an early age or to learn multiple programming languages right from the beginning, which is a major issue in designing computer-science curricula.

The involvement of working memory and attention may indicate that both should be trained during programming education. So, it is certainly worth exploring whether program comprehension can be improved by training specific cognitive abilities (e.g., through puzzle games). However, researchers disagree to what extent both can be learned or are rather inborn [22, 71, 78]. Thus, a test prior to programming education [48, 41] might reveal which students might struggle with learning programming. Especially when thinking of dyslectic students, who often have poorer short term memory and reading skills compared to non-dyslectic students [57], we may expect they struggle; however, many Dyslexics report that they can work focused during programming, for example, because of syntax highlighting [57]. Thus, unraveling the mind of dyslectic students might give us interesting insights into program comprehension.

Having found a strong involvement of language processing suggests that we need excellent language skills to become excellent programmers. Thus, if we loved learning new languages, we might also more easily learn new programming languages. It may be worthwhile to start learning a new (programming) language early during childhood, because studies showed that learning a second language early can have benefits regarding cognitive flexibility, metalinguistic, divergent thinking skills, and creativity [19]. Similarly, training *computational thinking*, a fundamental skill for computer scientists [77], prior to learning programming might also give novices a better start with learning programming, for example, to correctly specify unexpected states in a program [29].

Furthermore, despite similar education or experience, researchers have observed a significant gap between top developers and average developers, typically reported as a factor of 10 in terms of productivity [15, 17, 63]. However, nobody knows exactly how these top developers became top developers—they just are excellent. This raises many questions about to what extent we can train programmers at all. Alternatively, we can ask whether it is possible to predict whether somebody is inclined to become a great programmer. To answer such questions, we need to know how an excellent programmer differs from a normal programmer. Interestingly, characteristics of experts have been studied in many fields. For example, in an fMRI study, musicians showed a much lower activation in motor areas when executing hand tapping than non-musicians [37], and expert golfers, compared to novices, showed a considerably smaller activation pattern when imagining hitting a golf ball, because they have somewhat abstracted the activity [45]. In the same rein, excellent programmers may approach program comprehension differently. Again, understanding the differences may offer us insights into how to teach beginners, and, in the long run, develop guidelines for teaching programming.

Programming-Language Design. Traditionally, programming-language design does only marginally involve programmers and how they work with source code. Instead, experience and plausibility are used, such as: "As the world consists of objects, object-oriented programming is an intuitive way to program", "As recursion is counter intuitive, recursive algorithms are difficult to understand", or "Java (C#) shall be similar to C/C++ (Java), such that many developers can easily learn it." While experience and common sense are certainly valuable and may hint some direction on how to design programming languages, many design decisions that arise from them have—to the best of our knowledge—never been tested empirically (the work of Hanenberg is a notable exception [32]).

In our experiment, we have explored only small imperative code fragments with only few language constructs. It would be interesting to investigate whether there are fundamentally different activations when using more complex language constructs or using a functional or object-oriented style. For example, when we let developers understand object-oriented source code, we should observe activation in typical object-processing areas (e.g., BA 19 or 37), if real-world objects and object-oriented programming are similar, which is a frequently stated claim. The design of individual programming languages as well as entire paradigms may greatly benefit from insights about program comprehension gained by fMRI.

Furthermore, having identified a close similarity to language processing, we can further investigate how different or similar both processes really are. To this end, we envision letting participants read and comprehend natural-language descriptions as control task, instead of finding syntax errors; computing the difference in activation pattern, we can see how reading comprehension and program comprehension differ (if they differ at all). We also envision studies to explore the impact of *natural programming languages* [46] on comprehension, and how comprehension of spoken languages, dead languages (e.g., Latin) and programming languages differ.

Additionally, some researchers believe that the mother tongue influences how native speakers perceive the world (Sapir-Whorf hypothesis) [64, 76]. Since programming languages are typically based on English, Western cultures, compared to Asian cultures, might have a headstart when learning programming [4]. Taking a closer look at how developers from both cultures understand source code might give us valuable insights for teaching programming.

Software and Tool Design. Many questions regarding software design, modularity, and development tools arise. For instance, the typical approach to hierarchically decompose a software system leads to crosscutting concerns [73], but the extent to which developers naturally decompose a software system is unknown. Ostermann and others even argued that traditional notions of modularity assume a model based on classical logic that differs from how humans process information (e.g., they use inductive reasoning, closed-world reasoning, and default reasoning which are all unsound in classical logic) [50].

There has been considerable research in tool-based solutions for organizing and navigating software [27, 38, 39, 40, 61]. Considering navigation support, understanding how to support cognitive processes related to spatial abilities and to determine whether a given tool actually does support those abilities, might improve comprehension, provide a more disciplined framework for designing tools, and influence how we design software.

Software metrics. Many attempts have been made to measure the complexity of source code. Following initial proposals, such as McCabe's cyclomatic complexity [44] and Halstead's difficulty [31], a plethora of code and software metrics has been proposed [35]. Despite some success stories, it is still unclear why a certain metric works in a certain context and how to design a comprehensive and feasible set of metrics to assist software engineering. Which properties should a metric address? Syntactic properties, the control flow, the data flow, semantic dependencies, etc.? fMRI may give us a tool to answer these questions, for example, by analyzing whether complex data flows, for example, as targeted

by the DepDegree metric [8], give rise to distinct or stronger activations that correspond with complex comprehension activities.

9. RELATED WORK

We are not aware of any fMRI studies examining program comprehension. A few studies have used other physiological measures to study program comprehension. Hansen used eye-tracking to evaluate what parts of source code developers found most difficult [33]. Parnin used electromyography to measure the cognitive load during programming tasks [51].

In the neuroscience domain, several studies exist that also study tasks related to comprehension and detection of syntax errors. However, these studies, several of which were discussed in Section 5, use tasks involving only English words and sentences, not programs. The following studies are particularly interesting, because they revealed the same Brodmann areas as our study: In studies related to reading comprehension and language processing, participants had to understand text passages or decide whether sequences of letters can be produced with rules of a formal grammar [3, 9, 21, 24, 25, 26, 28, 30, 55, 56, 68, 74]. Regarding working memory, participants had to identify and apply rules or memorize verbal/numerical material [1, 47, 58, 69]. In divided-attention tasks, participants had to detect two features of objects at the same time [75].

Further work is needed to distinguish and dissociate brain activity related to program comprehension from other similar activities, such as word comprehension, and to allow us to develop a full model of program comprehension. Some researchers have already begun to theorize what a brain-based model of program comprehension would look like. Hansen and others propose to use the cognitive framework ACT-R to model program comprehension [34]. Parnin compiled a literature review of cognitive neuroscience and proposed a model for understanding different memory types and brain areas exercised by different types of programming tasks [52]. Both approaches are similar to our work by exploring knowledge of the neuroscience domain.

10. CONCLUSION

To shed light on the process of program comprehension, we used a relatively new technique: functional magnetic resonance imaging (fMRI). While in cognitive neuroscience, it has been used for more than 20 years now, we explored how fMRI can be applied to measure the complex cognitive process of comprehending source code. To this end, we selected twelve source-code snippets that participants should comprehend, which we contrasted with locating syntax errors.

We found that for comprehending source code, five different brain regions become activated, which are associated with working memory (BA 6, BA 40), attention (BA 6), and language processing (BA 21, BA 44, BA 47)—all fit well to our understanding of program comprehension. Our results indicate that, for learning programming, it may be beneficial to train also working memory, which is necessary for many tasks, and language skills, as Dijkstra already noted. We believe that fMRI has great potential for software-engineering research.

As a further contribution, we shared our experiences and design, to lower the barrier for further fMRI studies. We hope that fMRI becomes a standard research tool in software engineering, so that we can understand how developers understand source code and, eventually, tackle the really interesting questions: How do people use domain knowledge? To what extent is implementing source code a creative process? Can we train someone to become an excellent programmer? How should we design programming languages and tools for optimal developer support? Can software metrics predict the comprehensibility of source code?

Acknowledgments. We are grateful to all participants of our fMRI study and the pilot studies. Furthermore, we thank Jörg Liebig for his is support in conducting the first pilot study and Andreas Fügner for providing photos. Kästner's work partly by ERC grant #203099 and NSF award 1318808, and Apel's work by the DFG grants AP 206/2, AP 206/4, and AP206/5.

11. REFERENCES

[1] E. Awh, J. Jonides, E. Smith, E. Schumacher, R. Koeppe, and S. Katz. Dissociation of Storage and Rehearsal in Verbal Working Memory: Evidence from Positron Emission Tomography. *Psychological Science*, 7(1):25–31, 1996.

[2] A. Baddeley. Is Working Memory Still Working? *The American Psychologist*, 56(11):851–864, 2001.

[3] J. Bahlmann, R. Schubotz, and A. Friederici. Hierarchical Artificial Grammar Processing Engages Broca's Area. *NeuroImage*, 42(2):525–534, 2008.

[4] E. Baniassad and S. Fleissner. The Geography of Programming. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 510–520. ACM, 2006.

[5] Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.

[6] E. Berg. A Simple Objective Technique for Measuring Flexibility in Thinking. *Journal of General Psychology*, 39(1):15–22, 1948.

[7] A. Bethmann, C. Tempelmann, R. De Bleser, H. Scheich, and A. Brechmann. Determining Language Laterality by fMRI and Dichotic Listening. *Brain Research*, 1133(1):145–157, 2007.

[8] D. Beyer and A. Fararooy. A Simple and Effective Measure for Complex Low-Level Dependencies. In *Proc. Int'l Conf. Program Comprehension (ICPC)*, pages 80–83. IEEE, 2010.

[9] J. Binder, R. Desai, W. Graves, and L. Conant. Where Is the Semantic System? A Critical Review and Meta-Analysis of 120 Functional Neuroimaging Studies. *Cerebral Cortex*, 19(12):2767–2796, 2009.

[10] G. Bottini, R. Corcoran, R. Sterzi, E. Paulesu, P. Schenone, P. Scarpa, R. Frackowiak, and C. Frith. The Role of the Right Hemisphere in the Interpretation of Figurative Aspects of Language: A Positron Emission Tomography Activation Study. *Brain*, 117(6):1241–1253, 1994.

[11] K. Brodmann. *Brodmann's Localisation in the Cerebral Cortex*. Springer, 2006.

[12] R. Brooks. Using a Behavioral Theory of Program Comprehension in Software Engineering. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 196–201. IEEE, 1978.

[13] R. Brooks. Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. *Commun. ACM*, 23(4):207–213, 1980.

[14] R. Cabeza and L. Nyberg. Imaging Cognition II: An Empirical Review of 275 PET and fMRI Studies. *J. Cognitive Neuroscience*, 12(1):1–47, 2000.

[15] E. Chrysler. Some Basic Determinants of Computer Programming Productivity. *Commun. ACM*, 21(6):472–483, 1978.

[16] S. Cooper, W. Dann, and R. Pausch. Teaching Objects-First in Introductory Computer Science. pages 191–195. ACM, 2003.

[17] D. Darcy and M. Ma. Exploring Individual Characteristics and Programming Performance: Implications for Programmer Selection. In *Proc. Annual HawaiiInt'l Conf. on System Sciences (HICSS)*, page 314a. IEEE, 2005.

[18] B. de Alwis, G. Murphy, and S. Minto. Creating a Cognitive Metric of Programming Task Difficulty. In *Proc. Int'l Workshop Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 29–32. ACM, 2008.

[19] R. Diaz. Thought and Two Languages: The Impact of Bilingualism on Cognitive Development. *Review of Research in Education*, 10:23–54, 1983.

[20] E. Dijkstra. How do we Tell Truths that Might Hurt? In *Selected Writings on Computing: A Personal Perspective*, pages 129–131. Springer, 1982.

[21] N. Dronkers, D. Wilkins, R. Van Valin, Jr, B. Redfern, and J. Jaeger. Lesion Analysis of the Brain Areas Involved in Language Comprehension. *Cognition*, 92(1–2):145–177, 2004.

[22] W. Engle, J. Kane, and S. Tuholski. Individual Differences in Working Memory Capacity and what They Tell us about Controlled Attention, General Fluid Intelligence, and Functions of the Prefrontal Cortex. In *Models of Working Memory*, pages 102–134. Cambridge University Press, 1999.

[23] J. Feigenspan, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg. Measuring Programming Experience. In *Proc. Int'l Conf. Program Comprehension (ICPC)*, pages 73–82. IEEE, 2012.

[24] C. Fiebach, M. Schlesewsky, G. Lohmann, D. von Cramon, and A. Friederici. Revisiting the Role of Broca's Area in Sentence Processing: Syntactic Integration Versus Syntactic Working Memory. *Human Brain Mapping*, 24(2):79–91, 2005.

[25] A. Friederici. Towards a Neural Basis of Auditory Sentence Processing. *Trends in Cognitive Sciences*, 6(2):78–84, 2002.

[26] A. Friederici and S. Kotz. The Brain Basis of Syntactic Processes: Functional Imaging and Lesion Studies. *NeuroImage*, 20(1):S8–S17, 2003.

[27] W. Griswold, J. Yuan, and Y. Kato. Exploiting the Map Metaphor in a Tool for Software Evolution. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 265–274. IEEE, 2001.

[28] Y. Grodzinsky and A. Santi. The Battle for Broca's Region. *Trends in Cognitive Sciences*, 12(12):474–480, 2008.

[29] M. Guzdial. Education: Paving the Way for Computational Thinking. *Commun. ACM*, 51(8):25–27, 2008.

[30] P. Hagoort. On Broca, Brain, and Binding: A New Framework. *Trends in Cognitive Sciences*, 9(9):416–423, 2005.

[31] M. Halstead. *Elements of Software Science*. Elsevier Science Inc., 1977.

[32] S. Hanenberg, S. Kleinschmager, and M. Josupeit-Walter. Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code? An Empirical Study. In *Proc. Int'l Symposium Empirical Software Engineering and Measurement (ESEM)*, pages 156–167. IEEE, 2009.

[33] M. Hansen, R. Goldstone, and A. Lumsdaine. What Makes Code Hard to Understand? *ArXiv e-prints*, 2013.

[34] M. Hansen, A. Lumsdaine, and R. Goldstone. Cognitive architectures: a way forward for the psychology of programming. In *Proc. ACM Int'l Symposium on New Ideas, New paradigms, and Reflections on Programming and Software (Onward!)*, pages 27–38. ACM, 2012.

[35] B. Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, 1995.

[36] R. Hoge and G. Pike. Quantitive Measurement Using fMRI. In P. Jezzard, P. Matthews, and S. Smith, editors, *Functional Magnetic Resonance Imaging: An Introduction to Methods*, pages 159–174. Oxford University Press, 2001.

[37] L. Jäncke, N. Shah, and M. Peters. Cortical Activations in Primary and Secondary Motor Areas for Complex Bimanual Movements in Professional Pianists. *Cognitive Brain Research*, 10(1–2):177–183, 2000.

[38] C. Kästner, S. Apel, and M. Kuhlemann. Granularity in Software Product Lines. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 311–320. ACM, 2008.

[39] M. Kersten and G. Murphy. Mylar: A Degree-of-Interest Model for IDEs. In *Proc. Int'l Conf. Aspect-Oriented Software Development (AOSD)*, pages 159–168. ACM, 2005.

[40] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopez, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proc. Europ. Conf. Object-Oriented Programming (ECOOP)*, pages 220–242. Springer, 1997.

[41] T. Klingberg, H. Forssberg, and H. Westerberg. Training of Working Memory in Children With ADHD. *Journal of Clinical and Experimental Neuropsychology*, 24(6):781–791, 2002.

[42] S. Knecht, B. Dräger, M. Deppe, L. Bobe, H. Lohmann, A. Flöel, and E.-B. Ringelstein. Handedness and Hemispheric Language Dominance in Healthy Humans. *Brain*, 123(12):2512–2518, 2000.

[43] M. Knobelsdorf and R. Romeike. Creativity as a Pathway to Computer Science. In *Proc. Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, pages 286–290. ACM, 2008.

[44] T. McCabe. A Complexity Measure. *IEEE Trans. Softw. Eng.*, SE-2(4):308–320, 1976.

[45] J. Milton, A. Solodkin, P. Hlušítk, and S. Small. The Mind of Expert Motor Performance is Cool and Focused. *NeuroImage*, 35(2):804–813, 2007.

[46] B. Myers, J. Pane, and A. Ko. Natural Programming Languages and Environments. *Commun. ACM*, 47(9):47–52, Sept. 2004.

[47] Y. Nagahama, H. Fukuyama, H. Yamauchi, S. Matsuzaki, J. Konish, and H. S. J. Kimura. Cerebral Activation during Performance of a Card Sorting Test. *Brain*, 119(5):1667–1675, 1996.

[48] K. Oberauer, H.-M. Süß, R. Schulze, O. Wilhelm, and W. Wittmann. Working Memory Capacity—Facets of a Cognitive Ability Construct. *Personality and Individual Differences*, 29(6):1017–1045, 2000.

[49] R. Oldfield. The Assessment and Analysis of Handedness: The Edinburgh Inventory. *Neuropsychologia*, 9(1):97–113, 1971.

[50] K. Ostermann, P. Giarrusso, C. Kästner, and T. Rendel. Revisiting Information Hiding: Reflections on Classical and Nonclassical Modularity. In *Proc. Europ. Conf. Object-Oriented Programming (ECOOP)*, pages 155–178. Springer, 2011.

[51] C. Parnin. Subvocalization - Toward Hearing the Inner Thoughts of Developers. In *Proc. Int'l Conf. Program Comprehension (ICPC)*, pages 197–200. IEEE, 2011.

[52] C. Parnin and S. Rugaber. Programmer Information Needs after Memory Failure. In *Proc. Int'l Conf. Program Comprehension (ICPC)*, pages 123–132. IEEE, 2012.

[53] N. Pennington. Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs. *Cognitive Psychologys*, 19(3):295–341, 1987.

[54] S. Petersen, P. Fox, and M. Snyder, A.and Raichle. Activation of Extrastriate and Frontal Cortical Areas by Visual Words and Word-like Stimuli. *Science*, 249(4972):1041–1044, 1990.

[55] K. Petersson, V. Folia, and P. Hagoort. What Artificial Grammar Learning Reveals about the Neurobiology of Syntax. *Brain and Language*, 298(1089):199–209, 1982.

[56] K. Petersson and P. Hagoort. The Neurobiology of Syntax: Beyond String Sets. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* , 367:1971–1983, 2012.

[57] N. Powell, D. Moore, J. Gray, J. Finlay, and J. Reaney. Dyslexia and Learning Computer Programming. In *Proc. Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, pages 242–242. ACM, 2004.

[58] V. Prabhakaran, J. Smith, J. Desmond, G. Glover, and J. Gabrieli. Neural Substrates of Fluid Reasoning: An fMRI Study of Neocortical Activation During Performance of the Raven's Progressive Matrices Test. *Cognitive Psychology*, 33(1):43–63, 1996.

[59] C. Price, R. Wise, J. Watson, K. Patterson, D. Howard, and R. Frackowiak. Brain Activity During Reading: The Effects of Exposure Duration and Task. *Brain*, 117(6):1255–1269, 1994.

[60] J. Raven. Mental Tests Used in Genetic Studies: The Performances of Related Individuals in Tests Mainly Educative and Mainly Reproductive. Master's thesis, University of London, 1936.

[61] M. Robillard and G. Murphy. Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 406–416. ACM, 2002.

[62] A. Roskies, J. Fiez, D. Balota, M. Raichle, and S. Petersen. Task-Dependent Modulation of Regions in the Left Inferior Frontal Cortex During Semantic Processing. *J. Cognitive Neuroscience*, 13(6):829–843, 2001.

[63] H. Sackman, W. Erikson, and E. Grant. Exploratory Experimental Studies Comparing Online and Offline Programming Performance. *Commun. ACM*, 11(1):3–11, 1968.

[64] E. Sapir. *Culture, Language and Personality*. University of California Press, 1949.

[65] B. Shneiderman and R. Mayer. Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. *Int'l J. Parallel Programming*, 8(3):219–238, 1979.

[66] M. Shooman. The Teaching of Software Engineering. In *Proc. Technical Symposium on Computer Science Education (SIGCSE)*, pages 66–71. ACM, 1983.

[67] J. Siegmund, A. Brechmann, S. Apel, C. Kästner, J. Liebig, T. Leich, and G. Saake. Toward Measuring Program Comprehension with Functional Magnetic Resonance Imaging. In *Proc. Int'l Symposium Foundations of Software Engineering–New Ideas Track (FSE-NIER)*, pages 24:1–24:4. ACM, 2012.

[68] P. Skosnik, F. Mirza, D. Gitelman, T. Parrish, M. Mesulam, and P. Reber. Neural Correlates of Artificial Grammar Learning. *NeuroImage*, 17(3):1306–1314, 2008.

[69] E. Smith, J. Jonides, and R. Koeppe. Dissociating Verbal and Spatial Working Memory Using PET. *Cerebral Cortex*, 6(1):11–20, 1991.

[70] E. Soloway and K. Ehrlich. Empirical Studies of Programming Knowledge. *IEEE Trans. Softw. Eng.*, 10(5):595–609, 1984.

[71] D. Strayer. Driven to Distraction: Dual-Task Studies of Simulated Driving and Conversing on a Cellular Telephonen. *Psychological Science*, 12(6):462–466, 2001.

[72] J. Talairach and P. Tournoux. *Co-Planar Stereotaxic Atlas of the Human Brain*. Thieme, 1988.

[73] P. Tarr, H. Ossher, W. Harrison, and J. Stanley Sutton. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 107–119. ACM, 1999.

[74] A. Turken and N. Dronkers. The Neural Architecture of the Language Comprehension Network: Converging Evidence from Lesion and Connectivity Analyses. *Frontiers in Systems Neuroscience*, 5(1), 2011.

[75] R. Vandenberghe, J. Duncan, P. Dupont, R. Ward, J.-B. Poline, G. Bormans, J. Michiels, L. Mortelmans, and G. Orban. Attention to One or Two Features in Left or Right Visual Field: A Positron Emission Tomography Study. *J. Neuroscience*, 17(10):3739–3750, 1997.

[76] B. Whorf. *Language, Thought, and Reality*. Chapman and Hall, 1956.

[77] J. Wing. Computational Thinking. *Commun. ACM*, 49(3):33–35, 2006.

[78] S. Wootton and T. Horne. *Train Your Brain*. Teach Yourself, 2010.