A Look into Programmers' Heads

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

Abstract—Program comprehension is an important, but hard to measure cognitive process. This makes it difficult to provide suitable programming languages, tools, or coding conventions to support developers in their everyday work. Here, we explore whether *functional magnetic resonance imaging (fMRI)* is feasible for soundly measuring program comprehension. To this end, we observed 17 participants inside an fMRI scanner while they were comprehending source code. The results show a clear, distinct activation of five brain regions, which are related to working memory, attention, and language processing, which all fit well to our understanding of program comprehension. Furthermore, we found reduced activity in the default mode network, indicating the cognitive effort necessary for program comprehension. We also observed that familiarity with Java as underlying programming language reduced cognitive effort during program comprehension. To gain confidence in the results and the method, we replicated the study with 11 new participants and largely confirmed our findings. Our results encourage us and, hopefully, others to use fMRI to observe programmers and, in the long run, answer questions, such as: How should we train programmers? Can we train someone to become an excellent programmer? How effective are new languages and tools for program comprehension?

Index Terms—Functional magnetic resonance imaging, program comprehension

1 Introduction

Are learning natural languages and learning programming languages related? It may seem strange at first sight, but long ago, Dijkstra stated that "an exceptionally good mastery of one's native tongue is the most vital asset of a competent programmer" [22]. In Kentucky and several other US states, the legislative changed, such that school kids are now allowed to take a programming-language course instead of learning a foreign language.[1] Establishing a foundational theory of comprehension and cognitive processes associated with programming will fundamentally inform the design of software tools and programming languages as well as education policy for learning programming across multiple STEM disciplines. For example, we may be able to determine whether there is a critical age associated with programming skills. The results may also provide insights into how teaching programming practices, such as object-oriented languages, design patterns, or functional programming, are factors of how programmers comprehend code.

Research on the behavior of programmers has lead to interesting but limited insights in the context of program comprehension, the main activity of software developers. Theories of program comprehension have proposed two primary mechanisms: top-down and bottom-up program comprehension. *Top-down comprehension* is a hypothesis-driven process, in which developers initially form hypotheses about the source code and, by looking at more and more details, refine these hypotheses subsequently, until they form an understanding of the program [13]. With *bottom-up comprehension*, developers start with details of the source code and group these details to semantic chunks, until they have formed a high-level understanding of the program [96]. Thus, there are plausible models available that describe how developers proceed when understanding source code. However, the underlying cognitive processes of top-down and bottom-up program comprehension are still unclear—the programmer's head is still far from being understood.

To unravel the mysteries of program comprehension, we need to take a closer look at the underlying cognitive processes. Since *functional magnetic resonance imaging (fMRI)* has proved successful for observing internal cognitive processes, such as reading comprehension, concentration, object identification, and decision making, it is promising to apply it in the context of program-comprehension research.

In our experiment, 17 participants performed two types of tasks inside an fMRI scanner. In the first type, referred to as *comprehension tasks*, developers comprehended code snippets and identified the program's output. In the second type, 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. To gain confidence in the results and the method, we replicated the experiment with 11 new participants. As a result of our studies, we found:

- evidence that distinct cognitive processes took place when performing the comprehension tasks, as compared to the syntax tasks,
- activation of functional areas related to working memory, attention, and language comprehension, and
- a dominant activation of the left side, i.e., speech hemisphere.
- a relationship between source-code complexity and concentration level,
- a reduced cognitive effort with increased programming language familiarity, but

1. http://www.lrc.ky.gov/record/14rs/SB16.htm: KRS 156.160 Section 1.a.1: If a school offers American sign language or *computer programming language*, the courses shall be accepted as meeting the foreign language requirements *and the computer programming language course shall be accepted as an elective course* in common schools notwithstanding other provisions of law.

- no correlation between programming experience and cognitive effort.

Our results provide evidence of the involvement of working memory and language processing in program comprehension, and they imply that, during learning programming, training working memory (necessary for many cognitive tasks) and language skills (which Dijkstra already claimed as relevant for programming) might also be essential for programming skills. Furthermore, our results suggest that the more complex the source code is, the more the participants need to concentrate during comprehension tasks. Such results can help to validate or invalidate particular theories of program comprehension, such as data-flow measures that hypothesize an increasing need for concentration with a rising number of exchanged information between variables [8]. Although a single study is not sufficient to answer general questions, we can strengthen the confidence in our methodology and begin to ask probing questions and outline a path toward answering them: If program comprehension is linked to language comprehension, does learning and understanding a programming language require the same struggles and challenges as learning another natural language? If program comprehension dominantly activates the left hemisphere (often referred to as analytical), can we derive better guidelines on how to train students?

This article extends our paper presented at ICSE [99] and incorporates results from a replication performed as part of a different study [100]. Specifically, we

- expand our discussion and relation to other recent results, and discuss the history of and other alternative methods to fMRI (Section 2);
- show neural correlates of the effort participants put into comprehension task, and that the concentration level depends on the complexity of a source-code snippet (RQ2);
- show the effect of different levels of programming experience on a programmer's cognitive load (RQ3);
- show additional ways to analyze fMRI data (RQs 2 and 3).

With the extensions of the first fMRI study in the context of software engineering, this paper makes the following contributions:

- We present the first fMRI study to observe brain activity during bottom-up program comprehension tasks. By sharing our experience, we have already inspired other researchers to follow in our footsteps [15], [24], [33].
- We demonstrate the potential of fMRI studies to increase our understanding of the human factor in software-engineering research.
- We were able to successfully replicate our first fMRI study with different participants, which strengthens the validity of our experiment design [100].

Taking a broader perspective, our studies demonstrate the feasibility of using fMRI experiments in software-engineering research and has already led to other studies applying this technique [15], [24], [33]. With decreasing costs of fMRI studies, we believe that such studies will become a standard tool also in software-engineering research.

2 FMRI, Its Uses, and Limitations

To understand the principle of fMRI studies and why we designed our study the way we did, we shortly outline the development, limitations, and alternatives of fMRI.[2]

2. See Huettel et al. [48] for more details on the development of fMRI.

2.1 History

In the 1920s to the 1940s, researchers discovered that atomic nuclei have magnetic properties, which can be manipulated with magnetic fields. To this end, the magnetic field needs to oscillate with the resonant frequency of the atomic nuclei, which is referred to as nuclear magnetic resonance (NMR). In the following years, NMR was primarily used in chemistry to better understand chemical composition of homogeneous substances. In the 1970s, researchers discovered that water molecules behave differently in different biological tissues, leading to the hypothesis that these differences also occur in cancerous vs. non-cancerous cells. In studies with rat cells, this hypothesis could be confirmed, opening the door to medical application of NMR. In 1977, the first magnetic resonance image of a human's (a postdoctoral fellow) heart, lungs, and surrounding muscles was taken, which took 4 hours back then. NMR was a healthier alternative to computed tomography (CT), because it does not require exposure to concentrated X rays. In the early 1980s, NMR was renamed to MRI (magnetic resonance imaging) to remove the negative connotation of the word "nuclear" regarding health. Since then, it has been used in medicine to create structural images of the human body, which are used as diagnosis tool. The success in medicine ushered further research on MRI, which eventually led to *functional* MRI (fMRI), as we are using it in our study. In the 1990s, researchers observed that changes in blood oxygenation could be measured with MRI imaging. This is based on the different magnetic properties of oxygenated and deoxygenated blood. Oxygenated blood is diamagnetic and does not affect a magnetic field. Deoxygenated blood, however, is paramagnetic and does affect magnetic fields. This difference is exploited in fMRI by observing the BOLD signal.[2]

2.2 BOLD Signal

In a nutshell, fMRI observes magnetic properties of the blood. When a brain region is activated, its oxygen need increases, and so does the amount of oxygenated blood in this region. At the same time, the amount of deoxygenated blood decreases. Thus, the ratio of oxygenated and deoxygenated blood in a brain region changes compared to a resting state—this is referred to as the *BOLD (blood oxygenation level dependent)* effect. The BOLD effect needs a few seconds to manifest. Typically, about 5 seconds after stimulus onset, 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 briefly dips below the baseline [47]. A longer task duration allows the BOLD signal to accumulate, which produces better observed differences between tasks. To reliably measure the BOLD effect (i.e., the activation of a brain region), typical fMRI studies consist of an alternating sequence of task, control, and rest conditions without externally triggered mental activity.

2.3 Uses of fMRI Studies

When studying cognitive processes in the brain, scientists often follow a pattern of research that begins with a single case of brain injury that interferes with a cognitive behavior, followed by further studies locating and isolating brain activity. 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 which brain areas are involved in a given cognitive process, scientists further try identifying the timing and interaction of brain activity among brain regions.

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 [80].

To reference an identified brain location, Brodmann areas have proved useful as a classification system, and cognitive processes, such as seeing words or retrieving meaning from memory, can be mapped to these areas [12]. Through extensive research in this field over the past 25 years, there is a detailed and continuously growing map between Brodmann areas (and further subdivisions) and associated cognitive processes (e.g., www.cognitiveatlas.org shows an atlas).

When studying 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 6.1).

2.4 Limitations of fMRI

fMRI has now existed for more than 25 years and has provided valuable insights into the human brain [37]. However, fMRI also has limitations [83], [84]. First, the temporal resolution is limited, because changes in the BOLD contrast take longer than the underlying neural activity, which can occur as fast as 500 times per second [38]. We can currently tolerate this limitation, as many cognitive processes of interest to software engineering researchers (e.g., comprehension, maintenance, bug finding) are rather long-lasting (i.e., minutes). Temporal dynamics, for example, how the brain activation changes throughout a task, may be of interest once there is an established body of knowledge about cognitive processes and their activated brain areas. At this point, the limitation regarding temporal resolution will become relevant. However, while this kind of research is compelling, it is beyond the scope of this exploratory study to locate relevant brain areas of program comprehension.

Second, depending on the goal of a measurement, it requires averaging across a number of comparable events and participants to reduce unavoidable noise during data collection. The number of comparable events depends on the expected effect size: the larger the effect size, the fewer measurements are necessary. For example, for determining the speech hemisphere, the effect size is rather large, so that fMRI is capable of determining the speech hemisphere of individual participants within a few minutes [7]. To further increase temporal resolution, real-time approaches are currently developed that are able to determine the BOLD response of the whole brain online. However, determining subtle differences between very similar conditions still requires many averages as well as group data to determine meaningful and statistically significant results. Finally, we cannot make a causal inference, but only collect correlational data. That is, we can observe that a cognitive process occurs simultaneously with activation in a brain area, but we cannot conclude that this cognitive process *caused* the activation or that this activation *caused* the cognitive process.

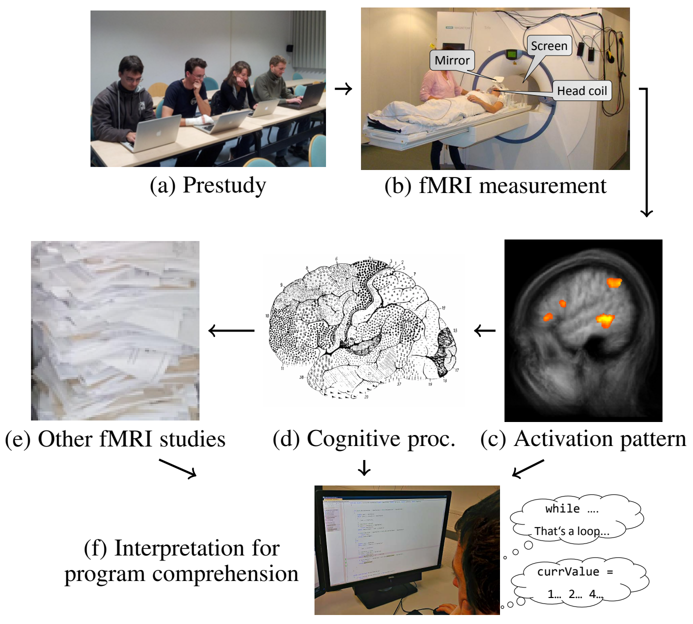


Fig. 1. Workflow of our fMRI study.

2.5 Alternatives

Since fMRI has a rather low temporal resolution and is currently quite expensive to conduct, other neuro-imaging techniques have also been used when observing cognitive processes (see Section 10 related studies).

Other techniques to measure brain activation also each have strengths and weaknesses. Electroencephalography (EEG) measures the electrical impulses of a neuron's activation and, thus, has a high temporal resolution, but at the cost of lacking the high spatial resolution of fMRI. Moreover, extracting event-related brain activation with EEG requires much more averaging than in fMRI. The same pertains to magnetoencephalography (MEG), which measures the magnetic properties of the electrical impulses of a neuron. Both techniques also have similar disadvantages in that they cannot collect signals from deep neural structures, but mainly from the surface of the brain. The same disadvantage holds for functional near-infrared spectroscopy (fNIRS). fNIRS also measures the BOLD effect, but the measurement is based on changes of light-absorbtion properties of oxygenated and deoxygenated blood [16]. However, an fNIRS device is much more light-weight than an fMRI scanner, making it much cheaper to apply.

3 FMRI Studies in a Nutshell

The sophisticated technical foundations of fMRI provide special challenges for the design of according studies. In the this section, we introduce the specific rationale of such studies on a high level. We describe the details of our study in Section 4.

3.1 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.

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 their head is fixed with cushions. Furthermore, communication and interaction with participants is limited, because speaking or pressing buttons may also cause motion artifacts. In such a restricted setting, the experiment duration should not exceed one hour, because after that, participants start getting restless or fatigued.

Additionally, participants can only view a relatively small screen reflected through a mirror (illustrated in Figure 1b), on which a limited number of lines of text can be displayed. We could support scrolling to show more text, but this is also prone to motion artifacts. This constraints us regarding the complexity of source code that we can present to participants. For example, studying how programmers comprehend an entire software component with many classes may increase motion artifacts due to the necessary actions to navigate the source code, and would also mean a high load on working memory, since not much can be shown on the screen. In the same vein, letting participants actually write code with a keyboard will be rather difficult, as participants are in a lying position, and because of the risk of elevated motion artifacts. Letting participants speak code is technically possible, but would cause additional language-related activation overlapping and diluting the activation of interest of program comprehension. Less restrictive techniques, such as fNIRS or EEG, are more suitable when it comes to observe writing code.

Finally, an fMRI study needs to be designed in a way to be able to distinguish brain activations caused by the experimental tasks from other non-specific activations. For example, 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 (e.g., visual processing for program comprehension), we need to design *control tasks* that allow us to compare brain activations between two conditions. The control tasks need to be as similar as possible to the experimental tasks and differ only in the absence of the targeted cognitive process. Ideally, the control tasks only differ in a single isolated aspect relevant to the research question, while keeping all other variables constant.[3]

3.2 Requirements for Our fMRI Study

With the goal of our study to find neuronal correlates of bottom-up program comprehension, the general fMRI challenges translate into a specific set of requirements.

First, we decided to start with short code snippets that fit on one screen to avoid motion artifacts by excessive usage of a computer mouse.

Second, we need source-code fragments with a suitable level of difficulty. If the source code is too easy to understand, participants may finish too early, such that the BOLD activation returns to 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. In a one-hour experiment, we can perform about a dozen repetitions, for which we need comparable tasks.

3. Amaro and Barker [1] give a detailed overview on the basics of fMRI study design.

Finally, as an fMRI study requires a control task to filter out irrelevant brain activation (cf. Section 3.1), we needed to find a task that ideally differs from the comprehension tasks 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 at 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 9. Results of fMRI studies can be generalized to realistic situations only with care.

3.3 Overview of Our fMRI Study

Given the constraints and our goal to observe bottom-up program comprehension, we selected short algorithms that are taught in first-year undergraduate computer-science courses as *comprehension tasks*, such as the word-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. We obfuscated identifiers to enforce program comprehension that required understanding code with a bottom-up approach, that is, from syntax to semantics (see Section 4.1).

For 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 the comprehension tasks (illustrated in Figure 3). Then, we asked participants to locate syntax errors (Lines 1, 2, and 8). Comprehension and syntax tasks are 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 (see Figure 1a). We let a total of 50 participants solve 23 comprehension tasks and search for more than 50 syntax errors. For the syntax-error tasks, we asked participants whether they needed to understand the source code in order to locate the errors, which occurred only occasionally. Based on our observations, we selected 12 source-code snippets and corresponding syntax errors with suitable duration and difficulty.

For the actual fMRI study (see Figure 1b), we conducted the experiment with 17 participants. Although initial fMRI studies often do not yield conclusive results because of missing empirical evidence (e.g., related studies, hypotheses about involved areas), we measured a clear activation pattern (Figure 1c), which is an encouraging result that we discuss in Section 6.

4 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.[4]

4. tinyurl.com/ProgramComprehensionAndfMRI/

```
public static void main(String[] args) {
  String word = "Hello";
  String result = new String();

  for (int j = word.length() - 1; j >= 0; j--)
  result = result + word.charAt(j);

  System.out.println(result);
}
```

Fig. 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 }
```

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

4.1 Objective

To the best of our knowledge, we performed the first fMRI study to measure program comprehension. Since we are exploring functional mappings of program comprehension a priori, we do not state research hypotheses about activated brain regions, but instead, pose three research questions about bottom-up program comprehension, code complexity, and programmer experience:

RQ1: Which brain regions are activated during bottom-up program comprehension?

We focused on bottom-up program comprehension to avoid any possible additional activation that is caused by participants relying on their domain knowledge. Recalling this knowledge might cause activation in memory-related areas, which could make the interpretation of the results more difficult. With bottom-up comprehension, we reduce such possible additional noise. Furthermore, we focused on a rather homogeneous level of programming experience and homogeneous difficulty of source-code snippets to reduce any noise caused by it. Of course, this higher internal validity of the design limits the external validity, which we discuss in Section 9.

The following research questions are intended to show the *potential* of fMRI and explore ways to maximize the output of expensive fMRI studies. To demonstrate this, we performed a secondary analysis of our data collected in our original study to answer two new research questions. However, we would like to note that, with our focus on internal validity (i.e., comparable source-code snippets and homogeneous level of programming experience), the results should be treated with caution, and the analysis should rather be seen as procedure for hypotheses generation.

RQ2: Does source-code complexity correlate with concentration levels during bottom-up program comprehension?

With increasing complexity of source code, it seems plausible that developers need a higher level of concentration to understand it. With fMRI, we can use the deactivation strength as an indicator of concentration levels [63]. Thus, with additional analysis, it is promising to explore this kind of question and evaluate its potential for future, dedicated studies.

RQ3: Does programming experience correlate with brain activation strength during bottom-up program comprehension?

Previous research showed that the brain activation (measured with EEG) differs for novices and experts [55], [59]. With fMRI, we can also differentiate between novices and experts for different activities (see Section 11). Thus, we show how we can look for such an effect in our fMRI study.

4.2 Operationalization of Variables

To answer RQ1, we need to ensure that participants use bottom-up comprehension only. At the same time, we did not want to increase the load on working memory unnecessarily. To balance the meaningfulness of identifier names, we conducted pilot studies (cf. Section 4.4), which showed that naming variables according to their purpose provides an optimal balance. For example, in Figure 2, variable `result` provides a hint of its purpose (i.e., that it contains the result), but does not reveal its content (i.e., that it holds the reversed word), and is also not completely unrelated (e.g., `aaaa`).

TABLE 1
Correlation values between the four chosen software measures for our twelve source-code snippets (cf. Table 2).

		Correlation
DepDegree	LOC	0.19
DepDegree	McCabe	0.44
DepDegree	Halstead	0.63
LOC	McCabe	0.41
LOC	Halstead	0.62
McCabe	Halstead	0.59

For RQ2, we need to find a measure of complexity for source code. Various software complexity measures have been proposed, which can be categorized into four groups:

- Size measures quantify the length of source code. The rationale is that, the more lines source code has, the more complex it is. We selected the commonly used LOC measure (without white spaces) as a representative measure [46].
- Control-flow measures quantify the complexity of the control flow of a program. The more possible execution paths exist, the higher the complexity. As representative, we selected McCabe’s cyclomatic complexity, which counts the number of possible execution paths [62].
- Another category of software measures aims at the vocabulary size of source code: The more variables and operations exist, the more cognitive effort is required to understand a program. As representative, we selected Halstead’s complexity [43].
- Data-flow measures quantify how much information is moved between program variables. The more information is moved, the more a developer has to consider and the more difficult it is to understand it, that is, where and what information is passed. We selected DepDegree as representative, which counts how often information is passed between program elements [8].

We selected one representative of each group, so that we do not miss potential interesting relationships between source-code complexity and concentration (i.e., to increase construct validity).

While there is a long-lasting discussion about the benefits and drawbacks of software measures, they are nevertheless used as indicator to assess the maintainability, extensibility, or comprehensibility of source code [30]. The benefit of software measures is that they are relatively easy to compute, but as drawback, their expressiveness for such aspects is unclear [27], [94], [107]. Additionally, with the plethora of software measures, it is not quite clear to what extent software measures actually describe different aspects of source code, or in other words, are independent. With our selection from different categories, we intend to mitigate this issue. However, the correlation of the software measures is quite high (see Table 1), except for DepDegree and LOC. Thus, the software measures are not independent for our source-code snippets. Nevertheless, we need to keep in mind that the snippets were not designed for this kind of analysis, and that the main purpose of conducting this analysis is to show a way to relate software measures to cognitive effort using fMRI.

To operationalize concentration, we evaluate changes in blood flow of the *default mode network*, which comprises several brain areas (e.g., cingulate cortex, prefrontal midline regions) and which is related to self-referential processing [37], [88]. When left to think about nothing specific (e.g., in the rest conditions), we often think about self-related aspects, for example, our plans for after the scanner session or previous experiences. This is reflected in an increased blood flow within the default mode network, that is, the default mode network shows high activation during rest states. When we concentrate on tasks, the default mode network *deactivates*, so that this self-referential processing does not interfere with the task. Hence, with the level of deactivation of the default mode network, we can measure the concentration level of participants: the stronger the deactivation, the higher the concentration level.

To address RQ3, we need to operationalize programming experience. To this end, we selected two measures:

- Program experience score as determined by our questionnaire (i.e., a combination of self-estimated experience with logic programming and self-estimated experience compared to class mates) [28].
- Java knowledge based on self estimation of the participants, because we used Java as underlying programming language.

4.3 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 warm-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. With this procedure, we minimized motion artifacts during the fMRI measurement. To ensure that comprehension took place, we showed the source code again directly after the scanner sessions, and participants entered their answer.

4.4 Material

Initially, we designed 23 standard algorithms that are typically taught in first-year undergraduate computer-science education at German universities. For example, we had algorithms for sorting or searching in arrays, string operations (cf. Figure 2), and simple integer arithmetic, such as computing a power function (see project's Web site 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 effect over 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).

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 can be located without understanding the execution of the program; they merely require some kind of pattern matching.

In a first pilot study [98], we determined whether the tasks have suitable difficulty and length. In a lab session, we asked participants to determine 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) to maximize the BOLD response in the fMRI scanner (cf. Section 3.2). Regarding correctness, we found that, on average, 90 % of the participants correctly determined the output, so none of the snippets had to be excluded 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 fMRI scanner, we further excluded four tasks to keep the experiment time within one 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 defined a fixed order for the source-code snippets to maximize the distance between snippets being shown for comprehension and locating syntax errors, while also preferably being shown as comprehension first. 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. This was not possible for three snippets (reverse array, cross sum, decimal to binary). With this fixed order, we attempted to minimize learning effects.[5] In Table 2, we give a high-level description of the source-code snippets, including the four different software complexity measures that we correlated with concentration.

Furthermore, we assessed the programming experience of participants with an empirically developed questionnaire to assure a homogeneous level of programming experience [28], and we assessed the handedness of our participants with the Edinburgh Handedness Inventory [72], because the handedness correlates with the role of the brain hemispheres [56] and, thus, is necessary to correctly analyze the activation patterns.

4.5 Participants

To recruit participants, we used message boards of the University of Magdeburg. We recruited 17 computer-science and mathematics students, two of them female, all with an undergraduate level of programming experience and Java experience (see project's Web site 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. While it seems counter-intuitive to select a homogeneous sample in terms of programming experience, the specific population will be programmers, and even novice programmers represent a relevant sample. In future studies, we will also recruit more experienced programmers.

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]. The participants gave written informed consent to the study, which was approved by the ethics committee of the University of Magdeburg. As compensation, the participants received 20 Euros. The participants were aware that they could end the experiment at any time.

4.6 Imaging Methods

The imaging methods are standard procedure of fMRI studies and are described in detail in this section.

4.6.1 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 $59\,\mathrm{cm}$, with a screen size of $325 \times 260\,\mathrm{mm}$, which is appropriate for an angle of $\pm 15°$. 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.

5. A randomized order, which is usually used to minimize learning effects, may have shown the same snippet as comprehension and locate syntax error directly after each other and as such not been effective for this study.

4.6.2 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 $1\,\mathrm{mm}$ 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 angle, $\pm 80°$; matrix size, 64×64; field of view, $19.2\,\mathrm{cm} \times 19.2\,\mathrm{cm}$; 33 slices of $3\,\mathrm{mm}$ thickness with $0.45\,\mathrm{mm}$ gaps). During the scans, participants wore earplugs for noise protection.

4.6.3 Data Preparation

We analyzed the functional data with BrainVoyager™ QX 2.1.2.[6] 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 [105].

Next, we spatially smoothed the functional data with a Gaussian filter (FWHM=4 mm). 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.

Additionally, we thoroughly inspected the functional data for strong signal intensity fluctuations resulting from head motion. For this purpose, we analyzed the automated head-motion-correction procedure, which resulted in estimated translation and rotation parameters for each spatial direction. In particular, we checked the data for jerky movements, as these can lead to signal artifacts. We defined a jerky move as a translation or rotation of the head from one volume to the next in the magnitude of $0.5\,\mathrm{mm}$ or $0.5°$ in one spatial direction or of $1.0\,\mathrm{mm}$ or $1.0°$ as the sum of all directions. We eliminated the respective volumes to correct for outliers.

4.6.4 Analysis Procedure

We conducted a random-effects GLM analysis, defining one predictor for the comprehension tasks and one for the syntax tasks. These were convolved with the model of a two-gamma haemodynamic response function using the default parameters implemented in BrainVoyager™ QX. We averaged the haemodynamic response for each condition (comprehension, syntax) across the repetitions.

Next, we contrasted comprehension with the rest condition using a significance level of $\mathrm{p} < 0.05$ (FDR-corrected [6]), to determine the voxels that showed a positive deflection of the BOLD response, compared to the rest period. This way, we were able to exclude voxels that were deactivated, which we analyzed separately. The activated voxels comprised a mask, which was used in the subsequent contrast, where we directly compared comprehension with syntax tasks at a significance level of $\mathrm{p} < 0.01$ (FDR-corrected) and a minimum cluster size of $64\,\mathrm{mm}^3$.

6. Brain Innovation B.V., Netherlands, brainvoyager.com

TABLE 2
Description of source-code snippets used in the study, ordered by the appearance of the comprehension task. One row represents the snippets of one trial. Column *Comprehension* denotes a snippet to be comprehended, column *Syntax* denotes the snippet of the same trial for which participants should locate syntax errors. The middle columns show the software measures of each comprehension snippet.

Comprehension	LOC	McCabe	Halstead	DepDegree	Syntax
Factorial	9	2	8.88	11	Decimal to binary
Check substring	15	3	15.28	14	Cross sum
Largest number	8	3	11.19	17	Reverse array
Reverse word	7	2	10.71	13	Check substring
Swap	9	1	6.77	5	Maximum in array
Power	9	2	10.40	12	Reverse word
Median	11	2	16.12	8	Factorial
Reverse array	11	3	18.35	25	Swap
Cross sum	9	2	11.50	11	Median
Largest of three numbers	11	7	15.86	15	Power
Count same chars	15	4	17.29	15	Largest of three numbers
Decimal to binary	12	3	14.00	16	Count same chars

To identify areas with task-induced deactivation within the default network, we contrasted program comprehension with the resting condition. To test whether there are systematic differences in task-induced deactivation elicited by the different program codes, we calculated for each of the resulting regions of interest the beta values from the general linear model for the 12 different code snippets. We then performed a two-sided Pearson correlation between these beta values and the different software measures, i.e., LOC, McCabe, Halstead, and DepDegree (RQ2). For RQ3, we computed the Spearman correlations of programming experience with the activation strength of the significant Brodmann areas.

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.

5 Results

In this section, we present the results, separated by the three research questions. For each research question, we first present the results, directly followed by the interpretation of the results. We discuss the implications of the results in Section 6.

5.1 RQ1: Which Brain Regions Are Activated during Bottom-Up Program Comprehension?

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 did not exclude any data from participants, since all showed comprehension of the source-code snippets by at least 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 Web site for details).

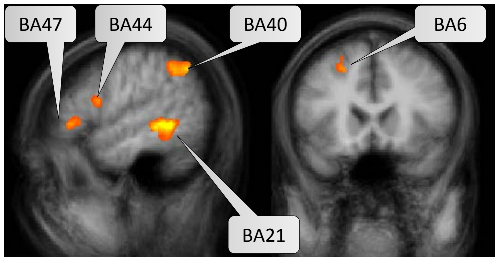
In essence, we found five relevant activation clusters, all in the left hemisphere. For each cluster, we show Talairach coordinates, the size of the cluster, related Brodmann areas, and relevant associated cognitive processes (note that deciding which cognitive processes are relevant belongs to the interpretation, not results; see Section 6). Thus, we can answer our first research question:

RQ1: During bottom-up program comprehension, Brodmann areas 6, 21, 40, 44, and 47 are activated.

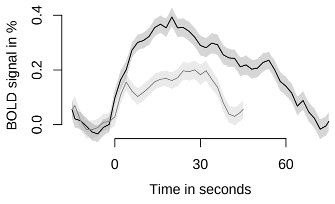
5.2 RQ2: Does Source-Code Complexity Correlate with Concentration Levels during Bottom-Up Program Comprehension?

In Figure 5, we show the significantly deactivated areas and their BOLD responses. The left part of Figure 5 shows the areas that are significantly deactivated during the comprehension task, that is, areas with less activation compared to the rest condition (color-coded with blue). The deactivated areas, that is, the prefrontal midline areas and posterior cingulate cortex, are both key components of the default mode network. The BOLD responses of the deactivated areas show a drop between 5 and 15 seconds after task onset. This indicates that participants concentrated during the comprehension tasks.

Next, we looked at the correlation between the strength of deactivation and complexity of source code for the set of software measures. Figure 6 visualizes the correlation of the software measures with the beta value for each deactivated areas averaged across participants. By using the mean, we can reduce the influence of peculiarities of individual participants. Each dot in the plot indicates one comprehension task. We found that all deactivated areas correlate negatively with DepDegree and Halstead; that is, the higher the value for these complexity measures, the lower the level of the beta value for the deactivated areas (indicating more concentration). One correlation, that is, the correlation of BA 32 with DepDegree (-0.591, bottom right) is statistically significant. Given our small sample size, we can actually expect that although some of the correlations have a high value, these are not necessarily statistically significant (cf. Section 4.1). Interestingly, the correlation with McCabe is positive, indicating that with a higher control-flow complexity, the deactivation of the found areas is less pronounced, or in other words, requires less concentration of participants. However, these correlations are not significant. The weakest correlations are with lines of code, indicating that there is no relationship between lines of code and concentration in our sample. However, it is important to note the source-code snippets were designed to be similar in length and complexity. Thus, while we demonstrated what such an analysis looks like, the source-code

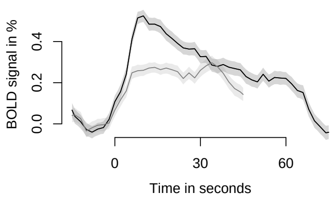


BA 6: Middle frontal gyrus
(Talairach coord.: -26, 17, 52; cluster size: 1279)



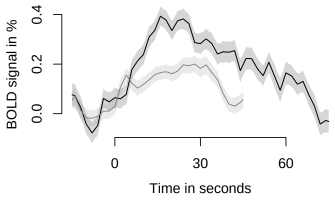
Attention
Division of attention
Language
Silent word reading
Working memory
Verbal/numeric
Problem solving

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



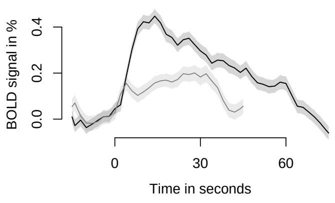
Semantic memory retrieval
Categorization

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



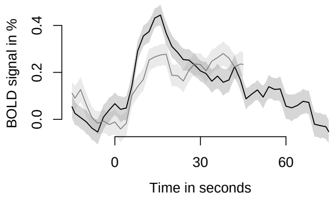
Working memory
Verbal/numeric
Problem solving

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

Fig. 4. Observed activation pattern for program comprehension and time courses of the BOLD response for each cluster. The time course in light gray is the BOLD response for the contrasting syntax condition. The shaded area around each time course depicts the standard deviation based on the participants. BA: Brodmann area.

snippets need to show a higher variation in length and complexity to reliably evaluate whether a relationship between concentration and length/complexity exists.

Another important issue to consider is the correlation of the software measures amongst themselves (cf. Table 1). For example, since Halstead and DepDegree exhibit a high correlation, it is only natural that both also have a high correlation with the deactivation strength. However, at the same time, Halstead also has a high correlation with LOC, yet LOC has almost no correlation with the deactivation strength. However, since we have a small sample and the snippets are not designed to actually provide an answer to this question, we are not digging deeper into explaining this result. The purpose of this analysis is to merely show how software measures can be related to cognitive effort.

5.3 RQ3: Does Programming Experience Correlate with Brain Activation Strength during Bottom-Up Program Comprehension?

We found five activated brain areas during program comprehension, which are shown in Figure 4. For RQ3, we computed the Spearman correlation between the *strength* of activation in the five activated brain areas during program comprehension and a participant's programming experience and Java knowledge (visualized in Figures 7 and 8).

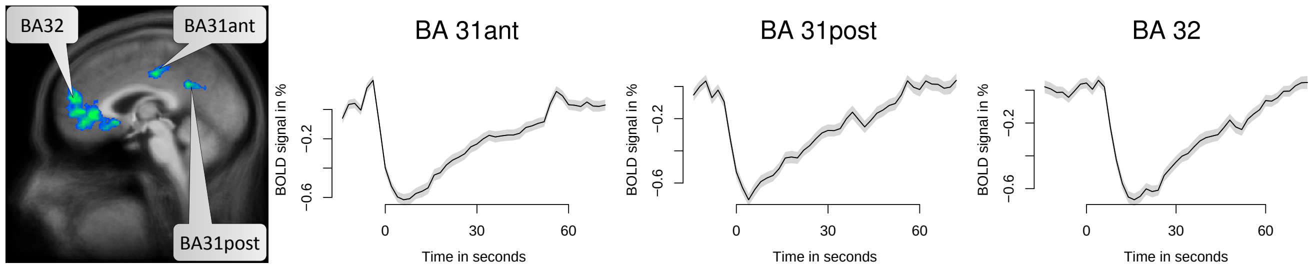
The distribution of programming experience scores is clustered around a low score value of 2.0. This can be explained by the homogeneous participant group of computer science students, which are rather novice programmers. Nevertheless, it is noteworthy that the correlations are different between the five brain areas. BA 21 shows a weak *positive* correlation (0.211). BA 6, BA 40, BA 44, and BA47 show a weak *negative* correlation (-0.089, -0.027, -0.202, and -0.148, respectively). However, the correlation between programming experience score and activation strength is not statistically significant for any of the five activated brain areas.

Self-estimated Java knowledge provides a more varied distribution. That means while the participants are overall rather inexperienced, their individual Java knowledge is diverse. The correlation between the Java knowledge and the activation strength is *negative* for all five activated brain areas. Hence, participants with more Java experience tend to have a lower activation strength. In other words, the data indicate that programmers familiar with Java require less cognitive effort to understand Java source code. In particular, BA 6 and BA 21 show a strong and significant negative correlation (-0.514, and -0.601, respectively). The activation strength of BA 40, BA 44, and BA 47 is also negatively correlated with the Java knowledge (-0.219, -0.379, and -0.21, respectively), but not statistically significant.

6 Discussion

6.1 RQ1: Which Brain Regions Are Activated during Bottom-Up Program Comprehension?

In our study, we found a distinct activation pattern of five Brodmann areas. In an initial study with such limited understanding of the role of different cognitive processes for bottom-up program comprehension, finding such a clear pattern is not the norm and demonstrates that the results (and our methodology) are very promising. To clarify the role of the activated brain areas and associated cognitive processes, we look at other fMRI studies that found the same areas activated as we did.



BA 31ant: Posterior cingulate cortex
(Talairach coord.: 0, -25, 40; cluster size: 1760)

BA 31post: Posterior cingulate cortex
(Talairach coord.: -3, -50, 28; cluster size: 1566)

BA 32: Anterior cingulate cortex
(Talairach coord.: -1, 40, 6; cluster size: 10919)

Fig. 5. Significant deactivation during program comprehension in the default mode network. The figure on the left visualizes the significantly deactivated areas and their location in the brain. The three graphs to the right show BOLD responses of each deactivated area. The deactivation slopes upwards after the peak, because some participants finish early. This effect is similar in the activated brain areas (cf. Figure 4).

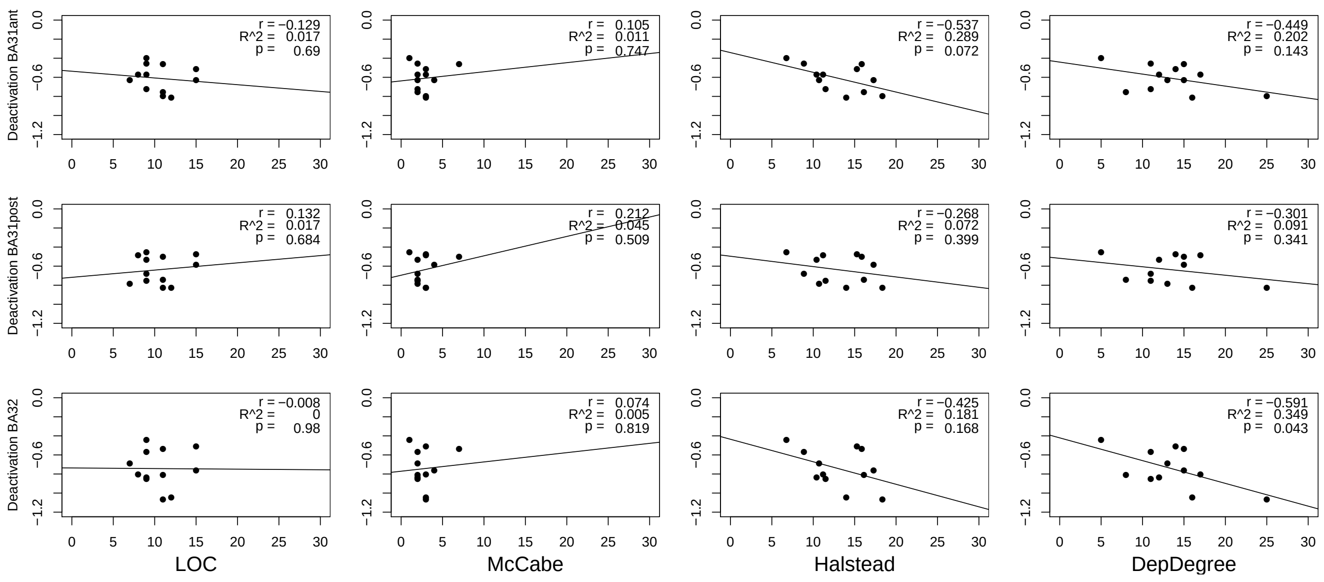


Fig. 6. Scatterplot of software measures and strength of deactivation of BA 31ant, BA 31post, and BA 32. Each dot represents one comprehension task. The strength of deactivation is the average beta value across all participants.

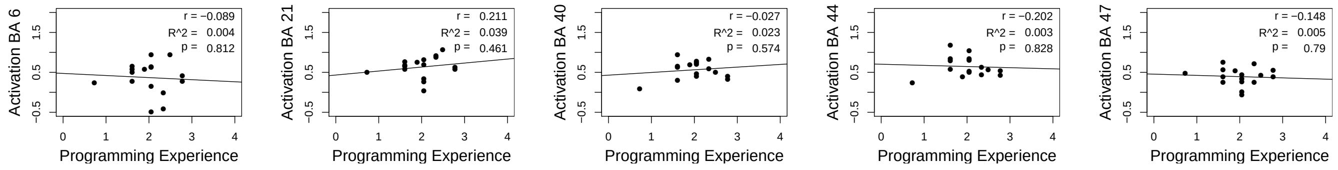


Fig. 7. Scatterplot of programming experience and activation strength of BA 6, BA 21, BA 40, BA 44, and BA 47. Each dot represents one participant. The strength of activation for each cluster is the average beta value across all snippets.

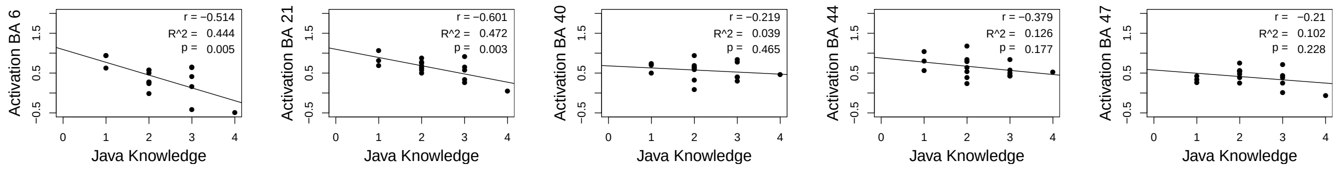


Fig. 8. Scatterplot of Java knowledge and activation strength of BA 6, BA 21, BA 40, BA 44, and BA 47. Each dot represents one participant. The strength of activation for each cluster is the average beta value across all snippets.

Brodmann areas 6 and 40 are often activated in cognitive tasks that require attention, working memory, or problem solving. In recent studies regarding typical problem-solving tasks for words and numbers (e.g., solving $4 + 1 = x - 1$ for x or $April + 1 = favorite - 1$ for $favorite$), BA 40 in the left inferior parietal cortex has been found to be consistently activated, indicating its important role in such activities [2], [69], [90]. Especially when applying algorithmic strategies to mathematical problems (e.g., for multiplying multi-digit numbers), the left inferior parietal cortex plays an important role [2], [90]. Its activation is modulated by the sophistication of strategies (e.g., the school strategy of multiplication from right to left vs. the expert strategy used in high-speed expert calculation, which goes from left to right), such that higher activation occurred in the posterior superior parietal lobule when applying the school strategy [90]. In our studies, participants also needed to solve problems based on numbers and words, that is, manipulating numbers or words in their mind according to the algorithms they identified in the source-code snippets. This also requires working memory in order to not forget the values of the manipulated words or numbers. Different studies consistently locate working memory, among others in BA 40 in the inferior parietal cortex [10], [11], [70], independent of whether numbers or words were manipulated. Common tasks that lead to activation in these areas include memorizing several numbers, consisting of pairs of numbers with mathematical structure (e.g., BAs 21, 32, 43, and 54) and without mathematical structure (e.g., BAs 18, 63, 90, and 47), or comprehending sentences with different syntactic structure that put different load on working memory. These kinds of tasks are comparable to comprehending source code, in particular regarding their working memory load.

Brodmann area 6 in the middle frontal gyrus is linked with tasks that require divided attention and is part of the attentional network [68], [102]. Like in our study, participants need to split their attention, for example, on two information streams and at the same time perform a one-back task (i.e., recall an item in a sequence of items that was shown before the current item) [68]. For comprehending source code, participants also need to keep in mind the value of variables and at the same time process control flow. These two processes each require attention.

What is also noteworthy here is that often, both BA 6 and BA 40 are found to be jointly activated for tasks that require high working-memory load [10], [11]. Especially for tasks, such as memorizing pairs of numbers or processing sentences with or without grammatical structure, which can pose a high load on working memory, a joint activation can be found, indicating that both areas are connected via neural pathways. In future studies, we will explore to what extent both areas in the frontal and parietal lobe play a joint or distinct role in bottom-up program comprehension.

In addition to other cognitive processes, 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 [4], [81], [101]. 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 are asked to 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 apply pattern matching in order 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 aspect of program comprehension 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 with accessing semantic information and the posterior MTG [9], [23], [108]. 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 BA 47) is related to combinatorial aspects in language processing, for example, processing of complex grammatical dependencies in sentences during syntactic processing [29], [40]. Several studies suggest that real-time combinatorial operations in the IFG incorporate the current state of processing and integrates incoming information into a new state of processing [42], [82]. Hence, the IFG was proposed to be involved in the unification of individual semantic features into an overall representation at the multi-word level [108]. 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.

While we discussed several activation clusters consistently found in studies on language comprehension and processing (i.e., BAs 21, 44, 47), we also need to discuss the fact that not the entire network that is associated with language processing was activated [87]. Specifically, we did not detect an activation in the anterior medial temporal lobe, which is commonly associated with semantic processing of language [87]. Similarly, we did not observe an activation in the left temporal pole, which is attributed to higher-level discourse processing [60]. In the same vein, additional areas for working memory might be observed in different studies, depending on the task (e.g., BAs 32, 43, and 54), indicating that not the entire network is recruited for program comprehension. Most likely, our tasks share not all cognitive sub processes of the other tasks reported in literature, indicating that the sub processes of program comprehension might be unique and tailored to comprehending source code. However, we need further studies to dissociate program comprehension and its sub processes from other cognitive processes.

In addition to the individual Brodmann areas, there is evidence for a direct interaction among 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 [29], [34], [35]. In contrast, BA 47 is assumed to serve as a semantic executive system that regulates and controls retrieval, selection, and evaluation of semantic information [91], [108]. 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 [108]. 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.

6.2 RQ2: Does Source-Code Complexity Correlate with Concentration Levels during Bottom-Up Program Comprehension?

With our rather small sample size and low number of snippets, the results are difficult to interpret. Nevertheless, we the analysis is very promising: The high correlation values with DepDegree and Halstead indicate that data-flow complexity and vocabulary size as operationalized by these measures modulate concentration levels of participants, which is in line with McKiernan's result of a stronger deactivation during more difficult tasks [63]. This also fits well to bottom-up comprehension, because there are no beacons to act as cues that could relieve cognitive load during comprehension. Instead, variable names remain rather abstract, and data-flow cannot easily be associated with certain variables.

In the past, many attempts have been made to measure the complexity of source code. For each complexity category from which we selected a representative measure (cf. Section 4.1), a plethora of code and software measures has been proposed [46]. However, it is still largely unclear why a certain measure works in a certain context and how to design a comprehensive and feasible set of measures to assist software engineering. Which properties should a code complexity measure address? Our data indicate that data-flow aspects and possibly vocabulary size affect concentration levels of participants, but not other aspects, such as syntactic properties or control flow. While not a definite answer, the hypothesis of a connection between data-flow complexity and vocabulary size, on the one hand, and required concentration of participants, on the other hand, is an intriguing hypothesis for future research, and our experimental design and analysis show a way to investigate this in more detail.

6.3 RQ3: Does Programming Experience Correlate with Brain Activation Strength during Bottom-Up Program Comprehension?

The missing correlation between programming experience (based on the experience score) and brain activation strength indicates that a higher general programming experience does not lead to a reduced cognitive effort. A follow-up study showed that top-down comprehension leads to a much lower activation strength (*neural efficiency*) than bottom-up comprehension [100]. However, our analysis here indicates that experienced programmers do not automatically show higher neural efficiency for a comprehension task in any programming language. Only experience in the specific programming language leads to a lower cognitive effort, as indicated by the negative correlation between knowledge of the Java programming language and brain activation. In other words, programming skills might not be efficiently transferred [79] to any domain, so an expert programmer might fall back to the neural efficiency of a novice when working with an unfamiliar language or domain. Floyd and others found a similar result in their fMRI study to analyze the difference in brain activation across programming-experience levels [33]. Their results show that program comprehension becomes increasingly similar to prose reading with higher programming experience.

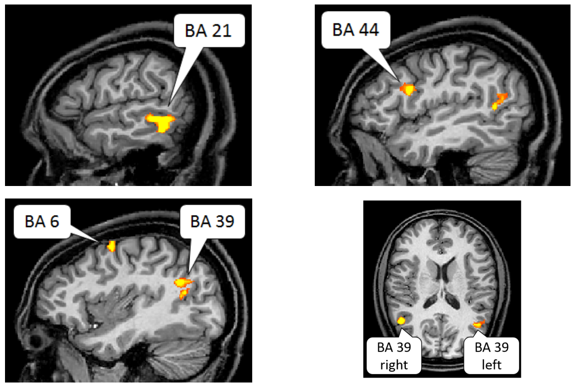


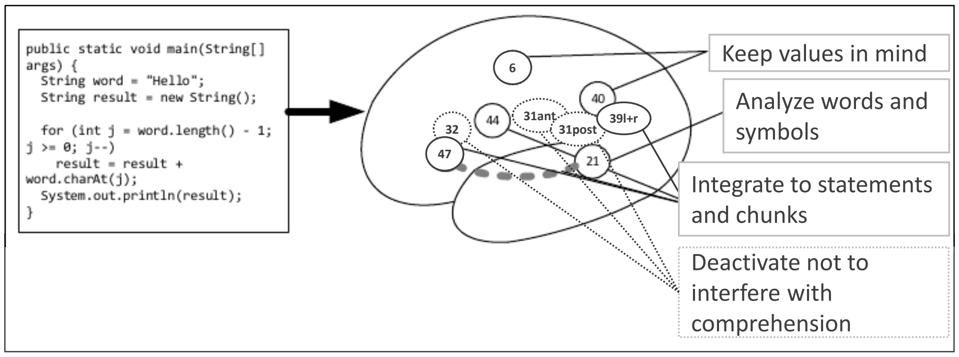
Fig. 9. Brodmann-area activation in replication study.

The strength of the correlations, especially for BAs 6 and 21, is surprising, as the experimental design was not primarily developed to address this question. This indicates that it is a very promising direction to further look into the role that the familiarity of a programming language plays for neural efficiency. The reduced activation strength in BA 21 indicates that being familiar with Java allowed our participants to be more efficient in analyzing the words and symbols of the source code. Consequently, the amount of values the participants had to keep in their working memory was reduced as well, which would explain the lower activation in BA 6. In conclusion, specific programming knowledge seems to have a strong effect on reading source code, and transferring knowledge between different programming languages is not that trivial. However, analyzing higher-level semantics, which is represented in BAs 40, 44, and 47, seems only moderately affected by familiarity with programming languages, indicating that this cognitive process is rather independent of the underlying programming language.

It is important to note that our interpretation of the data has to be treated with caution. The sample is small and rather homogeneous, and the strength of brain activation does not necessarily equal cognitive effort. To specifically examine cognitive effort, a parametric design or an independent measure (e.g., psycho-physiological parameters of EEG) is needed. Nevertheless, RQ3 shows the opportunities that fMRI provides for software-engineering research and how this kind of analysis can be used as hypothesis-generating procedure. Follow-up studies dedicated to this research question can build on our framework and a specialized design to fully understand the relationship between brain activation strength, cognitive effort, and programming experience. Can we identify expert programmers with fMRI? How transferable are programming skills between domains and languages? Our approach to fMRI data analysis gives researchers a new perspective and tool to answer these kinds of questions.

7 Replication Study

To gain confidence in the validity of our results for RQ1 regarding neural correlates of bottom-up program comprehension, we conducted a non-exact replication of our study. To this end, we made a few alterations, with the aim of increasing external validity (in



```
public static void main(String[]
args) {
  String word = "Hello";
  String result = new String();

  for (int j = word.length() - 1;
j >= 0; j--)
    result = result +
word.charAt(j);
  System.out.println(result);
}
```

6
40
44
31ant
31post
39l+r
32
47
21
Keep values in mind
Analyze words and symbols
Integrate to statements and chunks
Deactivate not to interfere with comprehension

Fig. 10. Visualization of how bottom-up program comprehension might take place.

the original paper [100], we present details on the study design). First, we added new code snippets as compared to the original version, but kept a similar level of length and complexity (e.g., finding the largest number in an array of numbers or double the entries in an array). To still be able to compare the results to our original study, we also kept a subset of the original snippets. On the supplementary Web site,[7] we show all the snippets that we used. Second, we shorten the comprehension condition to 30 seconds to reduce the chance of participants finishing a task and starting to rest early, which reduces contrast strength. Third, we randomized the snippet order to exclude learning effects. Lastly, we recruited 11 new participants who did not participate in the first study. The number of participants may seem low for conventional or other neuro-imaging methods. However, because program comprehension yields a high percentage signal change, the sample size is sufficiently large to find these effects [20]. To ensure comparability, we drew the sample from the same population. The participants' mean age was 25.3 ± 3.82 years, and all were familiar with Java or C at a medium level or higher.

The replication study was performed on a different MRI scanner (3 Tesla Philips dStream) using comparable measurement parameters. Because of the differences in experiment design and execution, we separately analyzed the data. We executed the same preprocessing and analysis steps, and found very promising results, such that essentially the same areas activated with a similar BOLD response (shown in Figure 9). Specifically, we could replicate the activation in BAs 21, 40, and 44. However, there are a few differences. First, we found that Brodmann area 39 was activated, which we did not find in the first study. We suspect one reason for missing this activation in the first study due to the small difference between comprehension and syntax-error finding, which we could not detect due to the rather low statistical power. In the replication, the activation was actually stronger than in the original study, so it was significant despite even smaller statistical power. BA 39 plays a crucial role as semantic hub to integrate multi-sensory information to form an understanding of events and solve familiar problems [95]. Thus, in future studies, we will specifically look at BA 39, even if it is not significantly activated, to further pinpoint the strength of the activation and identify its role for program comprehension. Second, we did not find an activation in BAs 6 and 47. For BA 6, it might be that subtle anatomical differences of participants might cause it to be attributed to different voxels. This assumption is strengthened by the fact that we found an activation, which can also be attributed to BA 6, in the same region close to the activation cluster of the original study. Mapping voxels to according Brodmann areas is not trivial [31], and it may be that subtle differences lead to a different mapping. For BA 47, the size of the area that the original study revealed was very small. In conjunction with the larger statistical power due to more participants and more comprehension tasks in the original study, this might explain why we could not replicate the activation of BA 47. Thus, to answer RQ1, we can state that BAs 6, 21, 40, and 44 are neuronal correlates of bottom-up program comprehension. BAs 39 and 47 also seem to play a role, but it is not as clear as for the other Brodmann areas.

Looking at the deactivated areas during the comprehension condition, we also found the same areas deactivated as in the original study, that is BA 31ant, BA 31post, and BA 32 (cf. Fig. 5).

We were not the only ones to replicate our study. Based on the same snippets, but with EEG, Lee and others found BAs 6 and 44 activated, indicating their consistent role during bottom-up program comprehension (see Section 10 for more details) [59]. Thus, the results our replication and of the study by Lee and others are very encouraging and strengthen the validity of our experimental design. They encourage us to keep using fMRI (and other neuro-imaging techniques) to better understand program comprehension and other related cognitive processes.

We did not analyze RQs 2 and 3 in the replication study, because the purpose of these RQs was to show the potential of fMRI. Although we increased external validity with the replication, we still did not design enough variability in the source-code snippets and sample to meaningfully answer these questions. In the future, we will further carefully increase external validity to also answer such questions.

8 Implications for Program Comprehension

Having identified activated and deactivated clusters and discussed their role for program comprehension in relation to other studies, we can now discuss the results from a higher level of abstraction. Specifically, we can hypothesize what a cognitive model of bottom-up program comprehension can look like. To understand source code, participants analyzed words and symbols and grouped them into semantic chunks. To this end, they need the language network of BAs 21, 44, and 47 that we found in our study. Additionally, BA 39 in both hemispheres might play a role during the integration. At the same time, participants

7. github.com/brains-on-code/paper-esec-fse-2017/

manipulate the values of numbers and words according to the intention of the source code. For this, they need to keep the values of the manipulated words and numbers in their mind, which is related to BAs 6 and 40. All parts, that is, semantic analysis of source code, manipulating the values of variables, and storing them in working memory, happens in parallel, for which participants need to divide their attention (located in BA 6). At the same time, with rising concentration levels, parts of the default mode network deactivated not to interfere with comprehending the source code. With increasing complexity of source-code snippets, the deactivation also gets stronger, indicating higher concentration levels. We illustrate this process in Figure 10.

Based on this model, we can hypothesize what influences program comprehension. For example, if we increase the number of variables beyond the working memory capacity of programmers, program comprehension should be impaired. Or, if we increase the complexity of source code, we might observe a parametrically stronger deactivation of the default mode network (more discussion in Section 11).

9 Threats to Validity

Inherent to our study design, there are threats to specific kinds of validity, which we discuss next.

9.1 Internal Validity

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 expertise from psychology, neurobiology, linguistics, as well as computer science and software engineering, for each process whether it might be related to our comprehension tasks. Additionally, all processes that are known to be associated with these Brodmann areas are available on the project's Web site.

9.2 External Validity

The source-code snippets that we selected were comparatively short, at most, 18 lines of code. Furthermore, we focused on bottom-up comprehension, and we explicitly avoided scrolling or typing to reduce any motion-related artifacts as far as possible. Thus, we focused on only one aspect of the complex comprehension process and cannot generalize our results to programming in the large—clearly, more studies have to follow. Nevertheless, it is conceptually possible to use a more complex setting, even tasks that last for several hours, and our results encourage us to try such larger settings in future studies, possibly also with other neuro-imaging techniques (see Section 2.5).

Another threat is that we kept the background of our participants, such as their programming experience and culture, constant to reduce the variability of the outcome. Furthermore, we did not control for gender of participants, which might bias the results, in that women show a tendency to prefer bottom-up comprehension [32]. Thus, we can generalize our results only carefully. In the next section, we outline, among others, how such personal differences might affect program comprehension.

9.3 Construct Validity

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 the time per source-code snippet was relatively short, we can assume that participants used bottom-up comprehension most of the time; this conjecture is supported by the fact that we did not observe activation in typical memory-related areas.

10 Related Work

10.1 Neuroscience

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 6, 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 [4], [9], [23], [29], [34], [35], [40], [42], [81], [82], [101], [108]. Regarding working memory, participants had to identify and apply rules or memorize verbal/numerical material [3], [66], [86], [103]. In divided-attention tasks, participants had to detect two features of objects at the same time [109].

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 [45]. 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 [75]. Both approaches are similar to our work by exploring knowledge of the neuroscience domain.

10.2 Software Engineering

With our fMRI study paving the way, other studies have followed. Specifically, Floyd and others conducted an fMRI study based on comprehension of source code and natural-language text [33]. In contrast to our work, the authors used the resulting activation pattern to predict the tasks that participants were completing. Duraes and others used fMRI to record activations during defect detection of software [15], [24]. The results indicate that initially finding and confirming a defect leads to different activation strength, especially in the right anterior insula.

Additionally, researchers are using other neuro-imaging techniques to observe programmers. Nakagawa and others used near-infrared spectroscopy (fNIRS) to measure changes in blood flow while programmers mentally executed source code. They found activation in the prefrontal cortex (a brain area that is necessary for higher-order cognitive processes), which correlated with the difficulty of a task [67]. In a similar study, Ikutani and Uwano found that activation in the frontal pole increases when participants memorized variables names, without manipulating their values [49].

Kluthe used electroencephalography (EEG) to measure program comprehension of participants with varying levels of expertise [55]. He let participants mentally execute the code and asked them to determine the output of source-code snippets. He found that, with lower expertise, program-comprehension tasks were more difficult to solve, indicating a higher cognitive load, which was reflected in the EEG signals. Lee and others used a similar experiment setup as we did, but also used EEG to record brain activation [59]. They found a subset of brain areas activated (i.e., Brodmann areas 6 and 44), confirming our findings on the role of these areas for bottom-up program comprehension.

In a follow-up study, Lee and others combined EEG with eye tracking to predict task difficulty and programmer expertise [58]. In this study, participants should comprehend source-code snippets similar to the ones in our study. In the same vein, we have conducted a further non-exact replication of our study and integrated eye tracking with the fMRI scanner [77], [78]. We could successfully map the eye movements of participants to their BOLD response. Fritz and others used three psycho-physiological measures—EEG, eye tracker, and electrodermal-activity sensor—to predict the difficulty of programming tasks [36]. Participants were required to mentally execute code that drew rectangles and decide whether rectangles overlap or determine the order in which rectangles were drawn. The authors found that these measures are promising to predict task difficulty. Fakhoury and others used functional near-infrared spectroscopy (fNIRS) and eye tracking to show how the quality of identifier names affects cognitive load [26]. They found that unsuitable identifier names increase cognitive load.

Thus, neuro-imaging studies are becoming more and more prevalent in software-engineering research.

11 Future directions

With our study, we show 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?

While our study provides only limited direct answers, it raises many interesting and substantial questions for future research: What are the underlying cognitive processes during top-down comprehension or the implementation of source code? How should we train programmers? How should we design programming languages and tools? Can software measures 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 an ongoing family of experiments, we are evaluating how participants use top-down comprehension and their memory to understand source code [100]. To this end, we show similar source-code snippets without obfuscating identifier names, and observe to what extent they serve as beacons for participants. Additionally, we ensure that participants are familiar with the source-code snippets. In this 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]. However, we did not find such an activation. By adding simultaneous eye tracking to our experiment framework [78], we have been evaluating whether participants fixate on beacons or familiar elements shorter or longer than unfamiliar statements, and how that gazing is related to neural activity [77]. Digging deeper in further studies, we may ask at which experience level beginners start using their previous knowledge? To what extent does the knowledge of the domain and other concepts, such as design patterns, influence activation patterns?

Measuring Complexity of Source Code: In the past, many attempts have been made to use software measures, such as code complexity [46], to understand why certain programming constructs or idioms may be more difficult to understand. Recent studies have tried to predict comprehensibility of source code based on a combination of software measures, but could only show small predictive power [94], [107]. All these approaches suffer from two main limitations: (1) software measures lack justification for cognitive outcomes; as a result, current software measures have poor predictive power on program comprehension [27], and (2) software measures are not explanatory, and cannot provide an answer to why certain constructs are more difficult than others.

Can we create cognition-based software measures to overcomes these limitations? We hypothesize that, in the future, such software measures will provide better predictions for assessing difficulty of source code.

Measuring Programmer Expertise: 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 [17], [19], [92]. 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 an excellent programmer [76].

To answer such questions, we need to know how an excellent programmer differs from an average 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 [50], 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 [64]. Another example are superior memorizers, who rely on specific strategies that build on visual information related to spatial landmarks when encoding large bits of information. Thus, compared to normal controls, superior memorizers recruit other brain regions, that is, those involved in visual processing and navigation [61].

In our study, we demonstrated how we can use brain activation strength, programming experience, and cognitive effort to better understand program comprehension. In the future and with dedicated studies, this may allow an objective assessment of programmer expertise beyond the standard skill tests and interviews.

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 BA 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 [74].

Training: There are many discussions about the best way to teach computer science and software engineering [18], [57], [97]. 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 often a controversial 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 [25], [104], [112]. Thus, a test prior to programming education [54], [71] might reveal which students might struggle with learning programming. Especially, when thinking of dyslexics, who often have poorer short-term memory and reading skills compared to non-dyslexics [85], we may expect they struggle; however, many dyslexics report that they can work with better focus during programming, because of syntax highlighting and other features [85]. Thus, unraveling the mind of dyslexics might give us interesting insights into program comprehension in general.

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 on during childhood, because studies showed that learning a second language early can have benefits regarding cognitive flexibility, metalinguistic, divergent thinking skills, and creativity [21]. Similarly, training *computational thinking*, a fundamental skill for computer scientists [111], prior to learning programming might also give novices a better start with learning programming, for example, to correctly specify unexpected states in a program [41].

Furthermore, excellent programmers may approach program comprehension differently. 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, designing programming languages only marginally involves empirical evidence of 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 shall be similar to C/C++, such that many developers can easily learn it." While experience and common sense are certainly valuable and may hint at some directions on how to design programming languages, many design decisions that arise from them have—to the best of our knowledge—only rarely been tested empirically (e.g., see Hanenberg [44]).

In our experiment, we have explored only small imperative-style 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 programming 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 are. To this end, we envision letting participants read and comprehend natural-language descriptions as control tasks, instead of finding syntax errors; computing the difference in activation patterns, we will see how reading comprehension and program comprehension differ (if they differ at all). We also envision studies to explore the influence of *natural programming languages* [65] on comprehension, and how comprehension of natural 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) [93], [110]. Since programming languages are typically based on English, Western cultures, as compared to Asian cultures, might have a headstart when learning programming [5]. 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 in software engineering. For instance, the typical approach to hierarchically decompose a software system is challenged by the presence of crosscutting concerns [106], but the extent to which developers naturally decompose a software system is unknown. Ostermann and others argued that traditional notions of modularity assume a model based on classical logic that differs from how humans process information (e.g., humans use inductive reasoning, closed-world reasoning, and default reasoning, which are all unsound in classical logic) [73]. Thus, we may need more natural concepts of modularity.

There has been considerable research in tool-based solutions for organizing and navigating software [39], [51], [52], [53], [89]. 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.

12 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 25 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 17 + 11 participants should comprehend, which we contrasted with locating syntax errors.

The key results are:

- A clear activation pattern of five different brain regions, 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 bottom-up program comprehension.
- A left-dominant activation, suggesting that language processing seems to be essential for program comprehension, which Dijkstra already noted [22]. With our study, we found first empirical evidence that Dijkstra may be right, which may have implications for teaching, such that training language skills, in addition to working memory and problem solving, might make programming education more efficient.

- An illustration of the potential of fMRI to develop cognitive-based complexity measures, and to relate programming experience and knowledge of the Java programming language to neural efficiency.
- A consistent result in two studies that reused our experimental design, indicating the validity of our experiment framework.

As a further contribution, our experience and methodology lowers the barrier for further fMRI studies. We hope that fMRI becomes a standard research tool in empirical software engineering, so that we and other researchers can understand how developers understand source code and refine existing models of program comprehension into a unified theory, so that we can eventually tackle the big questions in this area: 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 measures predict the comprehensibility of source code?

13 Acknowledgments

We thank all participants of both fMRI studies 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. Siegmund's and Brechmann's work is supported by DFG grants SI 2045/2-1 and BR 2267/7-1. Siegmund's work is further funded by the Bavarian State Ministry of Education, Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B). Apel's work has been supported by DFG grants AP 206/4, AP 206/5, and AP 206/6, and Kaestner's work by NSF awards 1318808 and 1552944, AFRL and DARPA (FA8750-16-2-0042), as well as in part by ERC grant #203099. Parnin's work is supported by NSF award 1755762.

References

[1] E. Amaro Jr and G. J. Barker. Study Design in fMRI: Basic Principles. *Brain and cognition*, 60(3):220–232, 2006.

[2] J. Anderson, S. Betts, J. Ferris, and J. Fincham. Cognitive and Metacognitive Activity in Mathematical Problem Solving: Prefrontal and Parietal Patterns. *Cognitive, Affective & Behavioral Neuroscience*, 11(1):52–67, 2011.

[3] 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.

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

[5] 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.

[6] 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.

[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] C. Bonhage, C. Fiebach, J. Bahlmann, and J. Mueller. Brain Signature of Working Memory for Sentence Structure: Enriched Encoding and Facilitated Maintenance. *Journal of Cognitive Neuroscience*, 26(8):1654–1671, 2014.

[11] D. Bor and A. Owen. A Common PrefrontalŰParietal Network for Mnemonic and Mathematical Recoding Strategies within Working Memory. *Cerebral Cortex*, 17(4):778–786, 2007.

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

[13] 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.

[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] J. Castelhano, I. C. Duarte, C. Ferreira, J. Duraes, H. Madeira, and M. Castelo-Branco. The Role of the Insula in Intuitive Expert Bug Detection in Computer Code: An fMRI Study. *Brain Imaging and Behavior*, May 2018.

[16] B. Chance, Z. Zhuang, C. UnAh, C. Alter, and L. L. Cognition-Activated Low-Frequency Modulation of Light Absorption in Human Brain. *Proc. Nat'l Academy Sciences of the United States of America (PNAS)*, 90(8):3770–3774, 1993.

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

[18] S. Cooper, W. Dann, and R. Pausch. Teaching Objects-First in Introductory Computer Science. In *Proc. Technical Symposium on Computer Science Education (SIGCSE)*, pages 191–195. ACM, 2003.

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

[20] J. E. Desmond and G. H. Glover. Estimating Sample Size in Functional MRI (fMRI) Neuroimaging Studies: Statistical Power Analyses. *Journal of neuroscience methods*, 118(2):115–128, 2002.

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

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

[23] 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.

[24] J. Duraes, H. Madeira, J. Castelhano, C. Duarte, and M. C. Branco. WAP: Understanding the Brain at Software Debugging. In *Proc. Int'l Symposium Software Reliability Engineering (ISSRE)*, pages 87–92. IEEE, 2016.

[25] R. Engle, M. 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.

[26] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope. The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load. In *Proc. Int'l Conf. Program Comprehension (ICPC)*, 2018.

[27] J. Feigenspan, S. Apel, J. Liebig, and C. Kästner. Exploring Software Measures to Assess Program Comprehension. In *Proc. Int'l Symposium Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. IEEE, 2011. paper 3.

[28] 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.

[29] 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.

[30] E. Figueiredo, C. Sant'Anna, A. Garcia, T. Bartolomei, W. Cazzola, and A. Marchetto. On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework. In *Proc. Europ. Conf. Software Maintenance and Reengineering (CSMR)*, pages 183–192. IEEE, 2008.

[31] B. Fischl, N. Rajendran, E. Busa, J. Augustinack, O. Hinds, B. Yeo, H. Mohlberg, K. Amunts, and K. Zilles. Cortical Folding Patterns and Predicting Cytoarchitecture. *Cerebral Cortex*, 18(8):1973–1980, 2008.

[32] M. Fisher, A. Cox, and L. Zhao. Using Sex Differences to Link Spatial Cognition and Program Comprehension. In *Proc. Int'l Conf. Software Maintenance (ICSM)*, pages 289–298. IEEE, 2006.

[33] B. Floyd, T. Santander, and W. Weimer. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 175–186, Piscataway, NJ, USA, 2017. IEEE Press.

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

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

[36] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, and M. Züger. Using Psycho-physiological Measures to Assess Task Difficulty in Software Development. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 402–413. ACM, 2014.

[37] M. S. Gazzaniga, R. B. Ivry, and G. R. Mangun. *Cognitive Neuroscience: The Biology of the Mind.* Norton & Company, 2013.

[38] B. Goldstein. *Sensation and Perception*. Cengage Learning Services, 5th edition edition, 2002.

[39] 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.

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

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

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

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

[44] 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.

[45] 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.

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

[47] 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.

[48] S. Huettel, A. Song, and G. McCarthy. *Functional Magnetic Resonance Imaging*. Sinauer Associates, 2008.

[49] Y. Ikutani and H. Uwano. Brain Activity Measurement during Program Comprehension with NIRS. In *IEEE/ACIS I.Í Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6. IEEE, 2014.

[50] L. Jäncke, J. 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] T. Kluthe. A Measurement of Programming Language Comprehension Using p-BCI: An Empirical Study on Phasic Changes in Alpha and Theta Brain Waves. Master's thesis, Southern Illinois University Edwardsville, 2014.

[56] 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.

[57] 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.

[58] S. Lee, D. Hooshyar, H. Ji, K. Nam, and H. Lim. Mining Biometric Data to Predict Programmer Expertise and Task Difficulty. *Cluster Computing*, pages 1–11, 2017.

[59] S. Lee, A. Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, and H. Lim. Comparing Programming Language Comprehension between Novice and Expert Programmers Using EEG Analysis. In *I.Í Conf. on Bioinformatics and Bioengineering (BIBE)*, pages 350–355. IEEE, 2016.

[60] E. A. Maguire, C. D. Frith, and R. Morris. The Functional Neuroanatomy of Comprehension and Memory: The Importance of Prior Knowledge. *Brain*, 122(10):1839–1850, 1999.

[61] J. Mallow, J. Bernarding, M. Luchtmann, A. Bethmann, and A. Brechmann. Superior Memorizers Employ Different Neural Networks for Encoding and Recall. *Frontiers in Systems Neuroscience*, 9(128), 2015. Published online.

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

[63] K. McKiernan, J. Kaufman, J. Kucera-Thompson, and J. Binder. A Parametric Manipulation of factors Affecting Task-Induced Deactivation in Functional Neuroimaging. *J. Cognitive Neuroscience*, 15(3):394–408, 2003.

[64] 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.

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

[66] 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.

[67] T. Nakagawa, Y. Kamei, H. Uwano, A. Monden, K. Matsumoto, and D. M. German. Quantifying Programmers' Mental Workload During Program Comprehension Based on Cerebral Blood Flow Measurement: A Controlled Experiment. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 448–451. ACM, 2014.

[68] K. Nebel, H. Wiese, P. Stude, A. de Greiff, H.-C. Diener, and M. Keidel. On the Neural Basis of Focused and Divided Attention. *Cognitive Brain Research*, 25(3):760–776, 2005.

[69] S. Newman, G. Willoughby, and B. Pruce. The Effect of Problem Structure on Problem-Solving: An fMRI Study of Word Versus Number Problems. *Brain Research*, 1410:77–88, 2011.

[70] S. Novais-Santos, J. Gee, M. Shah, V. Troiani, M. Work, and M. Grossman. Resolving Sentence Ambiguity with Planning and Working Memory Resources: Evidence from fMRI. *NeuroImage*, 37:361–378, 2007.

[71] 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.

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

[73] 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.

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

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

[76] C. Parnin, J. Siegmund, and N. Peitek. On the Nature of Programmer Expertise. In *Annual Workshop Psychology of Programming Interest Group (PPIG)*. PPIG, 2017.

[77] N. Peitek, J. Siegmund, C. Parnin, S. Apel, and A. Brechmann. Toward Conjoint Analysis of Simultaneous Eye-Tracking and fMRI Data for Program-Comprehension Studies. In *Proc. Int'l Workshop on Eye Movements in Programming*, pages 1:1–1:5. ACM, 2018.

[78] N. Peitek, J. Siegmund, C. Parnin, S. Apel, J. Hofmeister, and A. Brechmann. Simultaneous Measurement of Program Comprehension with fMRI and Eye Tracking: A Case Study. In *Proc. Int'l Symposium Empirical Software Engineering and Measurement (ESEM)*. ACM, 2018. To appear.

[79] D. N. Perkins, G. Salomon, and P. Press. Transfer of learning. In *International Encyclopedia of Education (2nd)*. Pergamon Press, 1992.

[80] 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.

[81] K. Petersson, V. Folia, and P. Hagoort. What Artificial Grammar Learning Reveals about the Neurobiology of Syntax. *Brain and Language*, 298(1089):199–209, 2012.

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

[83] R. Poldrack. Can Cognitive Processes Be Inferred from Neuroimaging Data? . *Trends in Cognitive Sciences*, 10(2):59–63, 2006.

[84] R. Poldrack. The Role of fMRI in Cognitive Neuroscience: Where Do We Stand? . *Current Opinion in Neurobiology*, 18(2):223–227, 2008.

[85] 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.

[86] 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.

[87] C. Price. A Review and Synthesis of the First 20 Years of PET and fMRI Studies of Heard Speech, Spoken Language and Reading. *NeuroImage*, 62(2):816–847, 2012.

[88] M. Raichle, A. MacLeod, A. Snyder, W. Powers, D. Gusnard, and G. Shulman. A Default Mode of Brain Function. *Proc. Nat'l Academy of Sciences*, 98(2):676–682, 2001.

[89] 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.

[90] M. Rosenberg-Lee, M. Lovett, and J. Anderson. Neural Correlates of Arithmetic Calculation Strategies. *Cognitive, Affective & Behavioral Neuroscience*, 9(3):270–285, 2009.

[91] 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.

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

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

[94] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Automatically Assessing Code Understandability: How Far Are We? In *Proc. Int'l Conf. Automated Software Engineering (ASE)*, pages 417–427. IEEE, 2017.

[95] M. L. Seghier. The Angular Gyrus: Multiple Functions and Multiple Subdivisions. *The Neuroscientist*, 19(1):43–61, 2013.

[96] 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.

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

[98] 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.

[99] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann. Understanding Understanding Source Code with Functional Magnetic Resonance Imaging. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 378–389. ACM, 2014.

[100] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, and A. Brechmann. Measuring Neural Efficiency of Program Comprehension. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 140–150, New York, NY, USA, 2017. ACM.

[101] 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.

[102] D. V. Smith, B. Davis, K. Niu, E. W. Healy, L. Bonilha, J. Fridriksson, P. S. Morgan, and C. Rorden. Spatial Attention Evokes Similar Activation Patterns for Visual and Auditory Stimuli. *J. Cognitive Neuroscience*, 22(2):347–361, 2010.

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

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

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

[106] 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.

[107] A. Trockman, K. Cates, M. Mozina, T. Nguyen, C. Kästner, and B. Vasilescu. "Automatically Assessing Code Understandability" Reanalyzed: Combined Metrics Matter. In *(MSR) Proc. Int'l Conf. Mining Software Repositories*, pages 314–318. ACM, 2018.

[108] 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.

[109] 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.

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

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

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

Norman Peitek is a PhD student at the University of Passau, Germany. He received his master's degree in Business Information Systems in 2014 from the University of Magdeburg. His research focuses on studying program comprehension with neuro-imaging methods.

Janet Siegmund is currently working at the University of Passau, where she is leading the junior research group PICCARD, funded by the Centre Digitisation.Bavaria. She received her Ph.D. from the University of Magdeburg in 2012 and she holds two master's degrees, one in Computer Science and one in Psychology. In her research, she focuses on the human factor in software engineering, for example, when writing source code. Janet Siegmund is the co-author of more than 30 peer-reviewed journal, conference, and workshop publications. She regularly serves as program-committee member for conferences and workshops. From 2014 to 2017, she was in the steering committee of the International Conference on Program Comprehension.

Sven Apel holds the Chair of Software Engineering at the University of Passau, Germany. The chair is funded by the esteemed Emmy-Noether and Heisenberg Programs of the German Research Foundation (DFG). Prof. Apel received his Ph.D. in Computer Science in 2007 from the University of Magdeburg, Germany. His research interests include software product lines, software analysis, optimization, and evolution, as well as empirical methods and the human factor in software engineering.

Christian Kästner is an assistant professor in the School of Computer Science at Carnegie Mellon University. He received his PhD in 2010 from the University of Magdeburg, Germany, for his work on virtual separation of concerns. For his dissertation he received the prestigious GI Dissertation Award. His research interests include correctness and understanding of systems with variability, including work on implementation mechanisms, tools, variability-aware analysis, type systems, feature interactions, empirical evaluations, and refactoring.

Chris Parnin is an assistant professor in North Carolina State University's Department of Computer Science. He received his PhD from the Georgia Institute of Technology and has published over 50 articles in software engineering and HCI.

Anja Bethmann received her diploma and PhD in linguistics from the Potsdam University in 2004 and 2012, respectively. Her research interest is in the neural mechanisms of semantic processing in the human brain as studied with functional MRI with a special focus on anterior temporal lobe function. Furthermore, she is interested in the rehabilitation of aphasia and active in the Saxony-Anhalt's patient organization on aphasia.

Thomas Leich received his diploma in Business Information Systems and his PhD from the University of Magdeburg in 2004 and 2012, respectively. Since 2013 he is general manager of the METOP GmbH. Since 2014 Thomas Leich is professor at the chair of Business Information Systems at Harz University of Applied Sciences. His research interests include requirements and software product-line engineering as well as measurement of program comprehension.

Gunter Saake is a full professor of Computer Science. He received his PhD in 1988 from University of Braunschweig. Currently, he is the head of the Databases and Software Engineering Group at the University of Magdeburg. His research interests include database integration, tailor-made data management, database management on new hardware, and feature-oriented software product lines.

André Brechmann is head of the Special-Lab Non-Invasive Brain Imaging at the Leibniz Institute (LIN) for Neurobiology Magdeburg since 2004. He received his diploma in biology from the University of Bielefeld in 1997 and his PhD in Neuroscience in 2002 from the Otto-von-Guericke University Magdeburg. In 2004 he was visiting researcher at the Martinos Center for Biomedical Imaging (MGH in Boston, USA) in preparation of the installation of Europe's first 7 Tesla MRI at LIN. Since 2012 he is coordinator of the Combinatorial NeuroImaging Core Facility (CNI) at LIN. His research interests include the dynamics and individuality of learning and memory processes in the human brain with a special focus on auditory cognition as well as other domains such as human-computer-interaction and program comprehension.