2017 IEEE/ACM 39th International Conference on Software Engineering

Decoding the representation of code in the brain: An fMRI study of code review and expertise

Benjamin Floyd
University of Virginia
bef2cj@virginia.edu

Tyler Santander
University of Virginia
ts7ar@virginia.edu

Westley Weimer
University of Virginia
weimer@virginia.edu

***Abstract*—Subjective judgments in software engineering tasks are of critical importance but can be difficult to study with conventional means. Medical imaging techniques hold the promise of relating cognition to physical activities and brain structures. In a controlled experiment involving 29 participants, we examine code comprehension, code review and prose review using functional magnetic resonance imaging. We find that the neural representations of programming languages vs. natural languages are distinct. We can classify which task a participant is undertaking based solely on brain activity (balanced accuracy 79%, $p < 0.001$). Further, we find that the same set of brain regions distinguish between code and prose (near-perfect correlation, $r = 0.99$, $p < 0.001$). Finally, we find that task distinctions are modulated by expertise, such that greater skill predicts a less differentiated neural representation ($r = -0.44$, $p = 0.016$) indicating that more skilled participants treat code and prose more similarly at a neural activation level.**

***Keywords*-medical imaging; code comprehension; prose review**

I. Introduction

Subjective human judgments are found in all stages of the software engineering lifecycle. Indeed, even when automated tools are available, humans must still choose to employ them. Because of their importance, many models and analyses have been proposed for such human judgments (e.g., of software readability [9], maintainability [26], or debugging tool output [55], etc.). The number of user evaluations and human studies at top venues has grown 500% since the year 2000 [10, Fig. 4]. Despite this, we have almost no understanding of how the human brain processes software engineering tasks. We propose to use medical imaging techniques to understand code comprehension and review.

Functional magnetic resonance imaging (fMRI) is a non-invasive technique for probing the neurobiological substrates of various cognitive functions *in vivo*. Technically, fMRI provides indirect estimates of brain activity, measuring metabolic changes in blood flow and oxygen consumption as a result of increased underlying neural activity. This signal is termed the blood-oxygen level dependent (BOLD) response [54]; it is formally defined as the ratio of deoxygenated to oxygenated hemoglobin, each of which have markedly different magnetic properties that can be detected by the MR-scanner (following excitation by a radio frequency pulse). This quantifies differences in activity under various conditions, both task-related and at rest. Functional MRI was first prototyped on humans in 1992, and has since enjoyed a meteoric rise in popularity among both clinical and psychological researchers. Unlike other cognitive neuroscience methods (e.g., EEG or PET), fMRI allows for rapid sampling of neural signal across the whole brain (1–2 seconds) and offers high spatial resolution (scale of millimeters) with regard to localizing signal sources. Thus, fMRI arguably provides the best available measure of online neural activity in the living, working human brain.

We present an fMRI study of software engineering activities. We focus on understanding code review, its relationship to natural language, and expertise. We note that the use of fMRI in software engineering is still exploratory; to the best of our knowledge this is only the second paper to do so [70], and is the first to consider code review and expertise. We explore these tasks because developers spend more time understanding code than any other activity [18], [29], [59], [62]. A NASA survey, for example, ranked understanding as more important than functional correctness when making use of software [53]. Similarly, with companies such as Facebook [77] and Google [36] mandating code review for new check-ins, code review has found an even greater practical and research prominence [85].

In our experiment, participants are presented with three types of visual stimuli, each with an associated judgment task. In the code comprehension task, participants are shown a snippet of code and asked a software maintenance question about it [71]. In the code review task, participants are shown a GitHub pull request (i.e., code, a patch to that code, and a comment) and asked whether they would accept it or not. In the prose review task, participants are shown English prose with simple editing markups and asked whether they would accept the proposed edits or not.

We designed our experiment in this manner because fMRI analyses are typically based on contrasts — comparisons between at least two task conditions. Through our experimental controls, we can isolate one feature of the tasks and observe differences in brain activation. For example, the primary difference between the prose review and code review tasks is whether the subject is English prose or a computer program; any difference in observed brain activity corresponds to a task difference. Similarly, a simple model might posit that code review is code comprehension followed by a code judgment (i.e., understanding what the patch will do and then deciding if it is acceptable); contrasting code review to code comprehension focuses on the judgment aspect, since code understanding is present in both. Mathematically, the analyses

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.24

we employed involved generalized linear models and Gaussian Process classification.

Ultimately, we find that the neural representations of programming and natural languages are distinct. We can construct classifiers that distinguish between these tasks based solely on brain activity. We observe that the same set of brain locations is relevant to distinguishing all three tasks. In addition, expertise matters: greater skill accompanies a less-differentiated neural representation.

The contributions of this paper are as follows:

1) We find that the neural representations of programming languages and natural languages are distinct. This claim is supported by a model that classifies participant tasks based on brain activity in a statistically significant manner (e.g., code comprehension vs. prose review at 79% balanced accuracy with $p < 0.001$).
2) We find that the same set of brain locations distinguish between all of these tasks. This claim is supported by regional importance maps (e.g., code vs. prose yields near-perfect correlation: $r = 0.99$, $p < 0.001$).
3) We find that the neural representations of programming languages and natural languages are modulated by expertise. Greater skill predicts less-differentiated representation. That is, expert brains treat code and prose tasks more similarly. This claim is supported by an analysis revealing a statistically significant correlation ($r = -0.44$, $p = 0.016$).
4) We make available our study materials and de-identified dataset of raw participant brain scans from 29 participants and discuss the barriers in conducting such a study.

II. Background and Motivation

In this section we present some relevant background on the software engineering tasks considered, as well as results related to expertise and imaging.

A. Code Review

Static program analysis methods aim to find defects in software and often focus on discovering those defects very early in the code's lifecyle. Code review is one of the most commonly deployed forms of static analysis in use today [73]; well-known companies such as Microsoft, Facebook, and Google employ code review on a regular basis [36], [77]. At its core, code review is the process of developers reviewing and evaluating source code. Typically, the reviewers are someone other than author of the code under inspection. Code review is often employed before newly-written code can be committed to a larger code base. Reviewers may be tasked to check for style and maintainability deficiencies as well as defects.

Numerous studies have affirmed that code review is one of the most effective quality assurance techniques in software development [1], [21], [25], [32]. While it is a relatively expensive practice due to high developer input, it is successful at identifying defects early in the development process. This benefit is valuable because the cost to fix a defect often increases with the time it goes unnoticed [79], [81], [84].

B. Code Comprehension

Much research, both recent and established, has argued that reading and comprehending code play a large role in software maintenance [30]. A well-known example is Knuth, who viewed this as essential to his notion of Literate Programming [39]. He argued that a program should be viewed as "a piece of literature, addressed to human beings" and that a readable program is "more robust, more portable, [and] more easily maintained." Knight and Myers argued that a source-level check for readability improves portability, maintainability and reusability and should thus be a first-class phase of software inspection [38]. Basili *et al.* showed that inspections guided by reading techniques are better at revealing defects [69]. An entire development phase aimed at improving readability was proposed by Elshoff and Marcotty, who observed that many commercial programs were unnecessarily difficult to read [24]. More recently, a 2012 survey of over 50 managers at Microsoft found that 90% of responders desire "understandability of code" as a software analytic feature, placing it among the top three in their survey [11, Fig. 4].

C. Expertise and Medical Imaging

As early as 1968, researchers began to measure productivity differences in computer science. Sackman *et al.* reported individual differences in programming performance (both in terms of programming and debugging person-hours to write the program and also in terms of the resulting CPU time taken by the program) of about an order of magnitude on average and up to 28:1 in some cases [63, Tab. 3]. While hardware performance can play a role, as in Doherty and Thadani's influential study [20], it does not explain all observed differences. More recent studies have explored the relationship between personality and performance in technical roles (e.g., [13]). While many studies report the impact of expertise on performance in various software engineering tasks (e.g., students with at most three years of experience are 18% less accurate at fault localization than are those with at least five [27, Tab. 2]), since medical imaging research is so new to software engineering, to the best of our knowledge there are no published results that attempt to relate performance differences to physical patterns of brain activation or anatomy.

Other fields have seen more direct inquiry. For example, Chi *et al.* examined the role of expertise in solving and classifying physics problems [15]. They report a 4:1 productivity difference between experts and novices, but also summarize methodological explanations: "both expert and novice proceed to solution by evoking the appropriate physics equations and then solving them. The expert often does this in one step, however" and "another interesting aspect of novice problem solving is not only that they commit more errors than experts but that, even when they do solve a physics problem correctly, their approach is quite different" [15]. They also describe a study in which participants classify physics problems. While experts categorize based on the underlying solution, novices focused on surface similarity (e.g., a pulley problem about force and a pulley problem about energy are similar to

novices). In their view, this suggests "that, with learning, there is a gradual shift in organization of knowledge" [15].

If so, how is this shift in knowledge organization reified in the brain? This question has been more thoroughly studied via medical imaging in the context of motor skills, where differences in functional plasticity differentiate novices and experts. Experts in a specialized motor skill show increased levels of activation in regions of the brain that control fine-tuned motor movements, suggesting that "progress from acquisition to automatization stages of motor skill learning is characterized by concomitant reduced demands on externally focused attention and executive function" [17]. Imaging studies have been conducted on monkeys, humans playing golf [61], and even humans juggling [65], finding similar physical explanations (e.g., at the level of the functional organization of neural networks during motor planning) of expertise. Maguire *et al.* found via fMRI that the navigation demands of being a London taxi driver stimulated brain development: drivers have larger-than-average relevant memory centers in their brains and the intensive training is responsible for that growth [46].

While psychology and neuroscience have made great strides using fMRI to relate and explain expertise in terms of the brain, no such investigation has been made for computing.

D. Computer Science and Medical Imaging

Siegmund *et al.* presented the first publication to use fMRI to study a computer science task: code comprehension [70]. The focus of their work was more on the novelty of the approach and the evaluation of code comprehension and not on controlled experiments comparing code and prose reasoning. Their analyses focused on standard generalized linear models to implicate relevant brain regions; by contrast we carry out a data-driven approach focusing on task classification and expertise. Our work was directly inspired by theirs.

Some researchers have begun to investigate the use of functional near-infrared spectroscopy (fNIRS) to measure programmer blood flow during code comprehension tasks. In smaller studies involving eleven or fewer participants, Nakagawa *et al.* found that programmers show higher cerebral blood flow when analyzing obfuscated code [52] and Ikutani and Uwano found higher blood flow when analyzing code that required memorizing variables [33]. fNIRS is a low-cost, non-invasive method to estimate regional brain activity, but it has relatively weak spatial resolution (only signals near the cortical surface can be measured); the information that can be gained from it is thus limited compared to fMRI. These studies provide additional evidence that medical imaging can be used to understand software engineering tasks, but neither of them consider code review or address expertise.

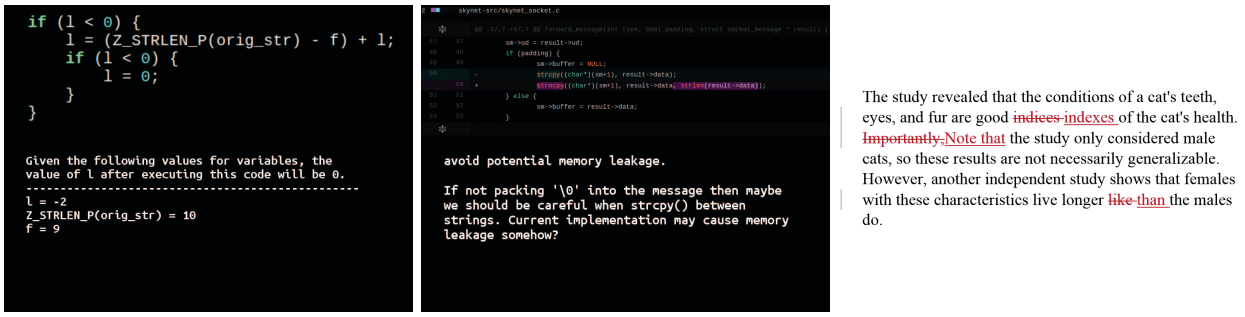
Understanding mental processing in computer science is important for a number of reasons: although this paper focuses on code comprehension, review and expertise, we briefly highlight six general motivations for pursuing medical imaging research related to computer science. First, to replace *unreliable self-reporting*. For example, in human studies of maintainability, three of the top four features humans self-report as relevant to their performance are actually irrelevant [26, Tab. 3]. The unreliability of self-reporting is not specific to computer science, and has been widely studied in psychology (e.g., [45], [56]). Medical imaging can give accurate, objective explanations of subjective processes. Second, to *inform pedagogy*. Medical imaging has already shown patterns of brain activation to predict learning rates [6] and memory tasks [60] in other domains. Even something as simple as whether certain activities are processed "like math" or "like language" by the brain would help provide foundational justification for certain introductory exercises. Third, to help *retrain aging engineers*. Medical imaging studies have found different patterns of activation across ages for other tasks. Older participants display more diffuse activation, recruiting nearby brain regions to help solve problems [12], [50]. Knowing whether or to what degree this holds true for engineering could guide workforce retraining or allocation, a growing issue (e.g., [82]). Fourth, to *guide technology transfer*. Since the decision to use a tool or not is made by a human, subjectivity matters (cf. [40]). For example, many developers avoid using tools with high false positive rates [35], but humans and tools may not agree on what is a false positive [4], [8]. As a concrete example, a number of fault localization approaches present their output as ranked lists, a form that humans dislike in this context [55]. Better models of how and why humans make such judgments would allow researchers to focus tool design (cf. [5, Sec. 3.4]). Fifth, to help *understand expertise*. Psychology and imaging studies have helped illuminate how brain structures change with expertise for other tasks, including chess [3], golf swings [61], and taxi driving [46]. Imaging may help us understand the long-reported order-of-magnitude productivity gap between experienced and novice programmers (e.g., [63]). Finally, and unabashedly, to provide foundational, *fundamental understanding*. For example, given that software is statistically akin to natural language [31], one may wonder how and when code is processed by the brain like language.

III. Experimental Setup and Method

In this section we describe our experimental protocol.

A. Participants

Thirty-five students at the University of Virginia were recruited for this study. Solicitations were made via fliers in two engineering buildings and brief presentations in a third-year "Advanced Software Development Techniques" class and a fourth-year "Language Design & Implementation" class. Data from six individuals were removed from the present analyses, either due to technical difficulties at the imaging center (yielding an incomplete dataset) or excessive head motion during the fMRI task. Thus, the final sample was comprised of 29 individuals (18 men, 11 women). Additional objective (e.g., GPA) and subjective (e.g., self-reported computing experience) information was gathered about each participant. Two of the participants were computer science graduate students, nine were undergraduates in the College of Arts and Sciences, and 18 were undergraduates in the College of Engineering. All



(a) Code Comprehension (b) Code Review (c) Prose Review

Fig. 1: Task stimuli. Code comprehension stimuli feature true and false claims in the style of Sillito *et al.* [71]. Code Review stimuli include the code difference (in color and with symbols) as well as the Git pull request message. Prose review stimuli are English paragraphs with proposed changes presented in a Microsoft Word "track changes" style.

participants were right-handed native English speakers, had normal or corrected-to-normal vision, and reported no history of neuropsychological disorder. They were also screened for basic experience in the programming language of interest. Prior to beginning the study, each individual provided written informed consent for a protocol approved by the University of Virginia Institutional Review Board (HSR #18240). Monetary compensation and course extra credit were offered.

B. Materials and Design

The experiment consisted of three tasks with unique stimulus sets: Code Review, Code Comprehension, and Prose Review. All stimuli were presented as images on a screen in the back of the MRI scanner, normalized to the native resolution of the projector (1024×768). Items were viewed through a mirror mounted atop the head coil.

C. Procedure

The full experimental protocol was completed over a single session per participant. Participants first viewed an instructional video detailing task requirements and were given further verbal instruction prior to entering the MRI scanner. Following an initial anatomical scan, participants completed four 11-minute runs of the code/prose task. In each run, stimuli were presented in alternating blocks of Code Review, Code Comprehension, and Prose Review; the blocks were ordered quasi-randomly across runs. All stimuli were presented for a fixed time (30 s for prose, 60 s for code) and required an Accept or Reject response, made on an MR-compatible button box held in the right hand. Participants were encouraged to respond as quickly and accurately as possible within the time allotted for each trial — neural responses were considered from the start of the trial until a decision was made. Inter-stimulus intervals ranged from 2–8 seconds and consisted of a white fixation cross displayed in the center of the screen. After completing the fMRI task, participants were given a chance to review the final run outside of the scanner and offer any verbal explanations for their responses.

D. Stimulus Type 1 — Code Comprehension

A code comprehension stimulus consists of a snippet of code and a candidate assertion about it (Figure 1a). Judging whether the assertion is true or not about the code requires comprehending the code. For comparability with previous research, we used the same code snippets as Fry *et al.* [26]. Some samples were reduced slightly in size to fit the fMRI projection screen and colors were inverted for readability. Nineteen total stimuli were used. The candidate questions were also taken from Fry *et al.*, and as thus ultimately adapted from Sillito *et al.*'s study of questions asked by actual programmers during software evolution tasks [71]. For each snippet a question type appropriate to the snippet was selected at random. Assertions were inducted from questions by including the correct answer or an incorrect answer (random coin flip). Assertions were used because the fMRI installation only allowed yes-or-no answers, not free-form responses.

E. Stimulus Type 2 — Code Review

A code review stimulus consists of an historical GitHub pull request, including the code difference and the developer comment (Figure 1b). Participants are asked to review the change and indicate whether they would accept it or not. A pool of candidate pull requests were selected by considering the top 100 C repositories on GitHub as of March 2016 and obtaining the 1,000 most recent pull requests from each. We considered the pull requests in a random order and filtered to consider only those with at most two edited files and at most 10 modified lines, as well as those with non-empty developer comments; the first twenty valid requests were used. Code was presented using the GitHub pull request web interface, simplified, and inverted for readability.

F. Stimulus Type 3 — Prose Review

A prose review stimulus consists of a snippet of English writing marked up with candidate edits (Figure 1c). Participants are asked to review the changes and indicate whether they would accept them or not. We included two sources of English writing. First, we selected random chapters from an

English writing textbook [22] that provides explicit correct and incorrect versions of candidate sentences and created examples based on grammar rules contained in those chapters. We created 20 stimuli of this type. Second, we selected random exercises in "Paragraph Improvement" from the College Board's SAT study guide [76]. Each such exercise has a paragraph and a series of questions about how to improve various parts of it; we created 10 stimuli by applying or reversing all of those changes. Prose review edits are shown via Microsoft Word "track changes".

IV. FMRI Analysis Approach

In this section we describe fMRI data capture and the mathematical analyses and modeling applied.

Since this sort of analysis is less common in software domains, we describe all steps in significant detail. However, our key areas of novelty are the overall experimental design, the use of binary Gaussian Process Classification to distinguish between code and prose tasks (Section IV-D), and our construction of interpretable weights for brain regions (Section IV-I). By contrast, the majority of individual steps taken (e.g., correcting for anatomy, denoising, cross-validation, etc.) are all well-established best practices for fMRI. In essence, this multi-step analysis is necessary because of the high dimensionality of medical imaging data, the timeseries nature of the tasks, the lag inherent in the BOLD signal, and our desire to both classify tasks and implicate brain regions.

A. Data Acquisition

We employed state-of-the-art imaging techniques using protocols obtained from the Human Connectome Project (HCP): a massive, multi-site effort to uncover the brain networks underlying human cognition [42]. These enabled rapid, parallel sampling of the BOLD signal across the whole brain, offering higher temporal resolution than standard protocols. They are also typically more robust to other artifacts that plague data quality (e.g., small head movements). MR signals are acquired in 3D *voxels* (volumetric pixels) that define a grid-like matrix over the brain. However, a full fMRI dataset is truly four dimensional (3D spatial + time).

All MR data were collected on a 3T Siemens Magnetom Tim Trio MRI system using a 32-channel head coil. Our functional MR scans employed a ${T_2}^*$-weighted multi-band echo planar imaging (mbEPI) sequence sensitive to the BOLD contrast (TR = 1 s; TE = 32 ms; FA = 90°, acceleration factor = 4). Whole-brain coverage was collected in 40 interleaved slices (3 mm slice thickness; 3×3 mm in-plane resolution). A total of 650 volumes were acquired for each of the four task runs. As a result of this process, a single participant completing four 11-minute runs produces 399,344,400 floating point numbers of data (153,594 voxels $\times$ 650 volumes $\times$ 4 runs). High-resolution anatomical images (strictly 3D, by contrast) were collected using a T_1-weighted magnetization-prepared rapid gradient-echo (MPRAGE) sequence (TR = 1.2 s; TE = 2.27 ms; FA = 9°; 176 slices; 1 mm thickness).

B. Data Preprocessing

Prior to any statistical analysis, fMRI data must be extensively *preprocessed*, which serves to correct systematic noise in the data (e.g., due to head motion) and align brains to a standard spatial template. This allows for straightforward comparison across subjects. Here, preprocessing and initial statistical modeling were performed using the Statistical Parametric Mapping 8 software [80] in Matlab. Standard preprocessing procedures were employed. First, functional data were realigned and unwarped to correct for participant head motion. High-res anatomical images were coregistered to the functional scans, all data were subsequently normalized to the Montreal Neurological Institute (MNI) template (cf. [47]). No spatial smoothing was applied to the functional data.

C. Generalized Linear Models

Following preprocessing, it is conventional to estimate within-subject general linear models (GLMs) to determine how the BOLD signal changes across various task-related events/conditions. Typically all trials for a given condition are collected in a single regressor, yielding an *average* BOLD response. However, here we isolated *unique* neural responses by estimating a single GLM per trial. This procedure, described by Mumford *et al.* [51], was necessary to avoid confounds related to hemodynamic lag when applying machine learning models to trials in an fMRI timeseries. For each GLM, one regressor modeled the BOLD response for the current trial of interest (event duration was curtailed at the participant's response time), and a nuisance regressor modeled all other trials within the run. The BOLD timeseries were high-pass filtered ($\sigma = 128$ s) and convolved with the canonical hemodynamic response function to estimate the unique neural response to each stimulus. As an additional denoising step, we estimated models using robust weighted least squares (rWLS [19]): this approach optimizes our measurement of trialwise BOLD responses by giving less weight to images with large noise variance. The end result of this process is a pseudo-timeseries of parameter estimate images (*beta images*), where each voxel's value describes the extent to which it is "activated" on a given trial (accounting for hemodynamic lag). These were subsequently used as training examples for within-subject machine learning models. "Missed" trials (i.e., where no response was given) were excluded.

D. Multivariate Pattern Analyses

We used Gaussian Process Classification (GPC) to determine the extent to which code and prose tasks elicited similar patterns of brain activity. If code and prose are processed using highly-overlapping brain systems, classifier accuracy would be low, reflecting entangled patterns of activity. These so-called *multivariate pattern analyses* were implemented in Matlab using the Gaussian Processes for Machine Learning software, v3.5 [58]. Classification is performed in a two-step procedure: the machine is first trained to identify patterns of activity corresponding to two stimulus types (code or prose),

and learning performance is then tested using new images without class labels.

E. Inputs and feature selection

The extremely large dimension of fMRI data is a major obstacle for machine learning — we commonly have tens of thousands of voxels but only a few dozen training examples. This can be solved using a simple linear map (a *kernel function*) that reduces the dimensionality of the feature space [43], [64], [68]. To begin, training inputs (*features*) were given as a set of vectors $\{\mathbf{x}_n\}_{n=1}^N$, with corresponding binary (+1/-1) class labels, $\{y_n\}_{n=1}^N$ (where N is the number of beta images for a given participant across both classes). Because any given beta image is a 3D matrix of voxels, we can easily reshape it into an input vector, $\mathbf{x}_n$.

The dimensionality of the feature vector is equal to the number of voxels used for pattern identification: for these analyses, we reduced the feature set to 47,187 voxels contained across 90 regions of the cerebrum, defined by the Automated Anatomical Labeling (AAL) atlas [78]. The AAL atlas allowed us to probe whole brain patterns across the same voxels for all participants. For additional feature reduction, we computed a simple $N \times N$ linear kernel whose elements indicated the degree of similarity between all pairs of input images.

F. Gaussian Process Classification

Gaussian Processes treat the classification problem as an extension of the multivariate Gaussian, defined by a covariance function that is used to make predictions for new data (conditioned on a training set). We elected to use GPC over other common methods (e.g., the support vector machine) for several reasons: 1) predictions are made by integrating over probability distributions (vs. hard linear decisions); 2) model hyperparameters and regularization terms are learned directly from the data (vs. costly nested cross-validation routines); and 3) maximum likelihood is robust to potential imbalances in class size, which otherwise bias linear classifiers toward predicting the more common class. GPs have also been successfully used in previous neuroimaging work to decode distinct cognitive states (as we do here), to distinguish healthy individuals from clinical populations, and even to predict subjective experiences of pain [14], [48], [67].

The technical minutiae of GPC analysis have been described in detail previously [41], [58]. Prior to training, a GP is defined entirely by its mean vector, μ, and covariance function, $\mathbf{K}$. The covariance function is parameterized as:

$$\mathbf{K} = \frac{1}{l^2}\mathbf{X}\mathbf{X}^T$$

where l^2 is a learned scaling parameter and $\mathbf{X}\mathbf{X}^T$ gives the linear kernel. The goal of GP-based machine learning is then to identify optimal covariance parameters that allow for accurate predictions of new data. However, because binary classification is by nature non-Gaussian (all $y \in \{+1, -1\}$), we adopt a *function space* view of GPs that models a latent distribution over functions, $f(\mathbf{x})$, given the data, $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$. This distribution is used to estimate relationships between the training data and make predictions for new examples.

To learn such a mapping, we employ a cumulative Gaussian (or *probit*) likelihood and specify the posterior conditional over $\mathbf{f}$ using Bayes' rule:

$$p(\mathbf{f}|\mathcal{D}, \theta) = \frac{\mathcal{N}(\mathbf{f}|0, \mathbf{K})}{p(\mathcal{D}|\theta)} \prod_{n=1}^{N} \phi(y_n f_n)$$

where $\mathcal{N}(\mathbf{f}|0, \mathbf{K})$ is a zero-mean prior, $\phi(y_n f_n)$ is a factorization of the likelihood over training examples, and $p(\mathcal{D}|\theta)$ gives the model evidence (or the marginal likelihood of the data given a vector of hyperparameters, θ). Training therefore involves finding the optimal form of $\mathbf{K}$ by scaling model hyperparameters and maximizing the (log) model evidence.

G. Expectation Propagation

Class predictions for new images were made using *expectation propagation* (EP). This was necessary because the probit likelihood and the posterior are both non-Gaussian, making exact inference analytically intractable. EP algorithms allow us to reformulate the posterior as a Gaussian and approximate the distribution of the latent function at a new test point, $\mathbf{x}_*$:

$$\begin{aligned} p(y_* = +1|\mathcal{D}, \theta, \mathbf{x}_*) &= \int \theta(f_*) q(f_*|\mathcal{D}, \theta, \mathbf{x}_*) df_* \\ &= \phi\left(\frac{\mu_*}{\sqrt{1+\sigma_*^2}}\right) \end{aligned}$$

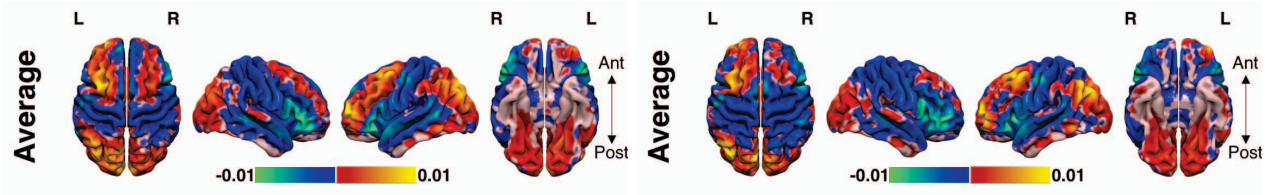
where $q(f_*|\mathcal{D}, \theta, \mathbf{x}_*)$ gives the EP approximation to a Gaussian. Importantly, we still obtain a true probabilistic inference by integrating over the latent posterior. The obtained class probability is converted to a binary class label by inverting the logarithm:

$$t_* = e^p \begin{cases} t_* > 0.50, & y_* = +1 \\ t_* \leq 0.50, & y_* = -1 \end{cases}$$

The 0.50 threshold is non-arbitrary, owed to the symmetry of the cumulative Gaussian.

H. Testing and Training

We mitigated overfitting via careful cross validation and estimated unbiased measures of classification performance. Together, these offered a robust means of testing the extent to which GPC could distinguish between code and prose-related patterns of activity. Ultimately three binary GPC models were trained and tested for each participant: Code Review vs. Prose Review, Code Comprehension vs. Prose Review, and Code Review vs. Code Comprehension. Predictive performance was assessed using a leave-one-run-out cross validation (LORO-CV) procedure. For each fold of LORO-CV, the data from one scanning run were removed from the kernel. The kernel was then centered according to the remaining training examples, the model was fit, and class predictions were made for the left-out data. Given that all participants did not necessarily have equal numbers of code/prose examples, average performance across all CV folds was estimated as the balanced accuracy (*BAC*), or the arithmetic mean of the two class accuracies.



(a) Code Comprehension vs. Prose Review

(b) Code Review vs. Prose Review

Fig. 2: Average weight maps for task classifiers. When regions of the brain colored "hot" are active, the decision is pushed toward Code. The left and right subfigures show a high degree of concordance ($r = 0.75$, $p < .001$), quantifying how both code tasks are distinguished similarly compared to the prose task.

I. Regional Inference

We next sought to determine which regions of the brain were most involved in discriminating between code and prose. This involved projecting kernel weights back onto the 3D brain — for display purposes, we present weight maps that were averaged across CV folds and participants. It is worth emphasizing, however, that such multivariate maps do not lend themselves to simple regional inference: because the final classification decision depends on information across *all* voxels, it is incorrect to assume voxels with high weight are the "most important." Nevertheless, we may estimate *a posteriori* the total contribution of each anatomical area in the aforementioned AAL atlas [78]. In this procedure, the absolute values of all voxel weights within a brain region were summed and divided by the total number of voxels in the region. Then, each region's "contribution strength" was divided by the sum of strengths for all regions, yielding a proportion that is directly interpretable as regional importance — a larger value indicates more total weight represented within a region [66]. These importance maps are also presented as a group average.

V. Results and Analysis

In this section we present our experimental results, focusing on the analysis of the raw participant data obtained from the experimental protocol in Section III. We focus our analysis on three research questions:

RQ1 Can we classify which task a participant is undertaking based on patterns of brain activation?
RQ2 Can we relate tasks to brain regions?
RQ3 Can we relate expertise to classification accuracy?

A. RQ1 — Task Classification

We assess if our learned models can classify which task a participant is performing based solely on patterns of brain activity. We consider the three tasks (Code Review, Code Comprehension and Prose Review) pairwise.

Since not all participants completed the same number of trials (e.g., some participants completed more prose trials than others), we first sought to ensure that this difference was not biasing classifier accuracy. Median *BAC* was compared for each of the three models using nonparametric Wilcoxon rank-sum tests [2]. There were no significant differences in classification performance for either Review vs. Prose models ($Z = 0.84$, $p = .400$) or Comprehension vs. Prose models ($Z = 0.55$, $p = .579$), suggesting that GPC's ability to discriminate between code and prose tasks was not driven by the number of prose trials completed. This also held when considering only Prose class accuracy in both Review vs. Prose models ($Z = -1.87, p = .061$) and Comprehension vs. Prose models ($Z = -1.53, p = .127$). A full set of summary statistics for classifier performance are displayed in Table I.

With regard to overall classifier performance, we employed nonparametric Wilcoxon signed-rank tests to compare model *BAC* against a null median accuracy of 50% (chance for a binary classifier). For all models, GPC performance was highly significant. The classifiers accurately discriminated between Review vs. Prose trials ($BAC = 70.83\%$; $Z = 4.00$, $p < .001$), Comprehension vs. Prose trials ($BAC = 79.17\%$; $Z = 4.51$, $p < .001$), and even Review vs. Comprehension trials ($BAC = 61.74\%$; $Z = 3.45$, $p < .001$).

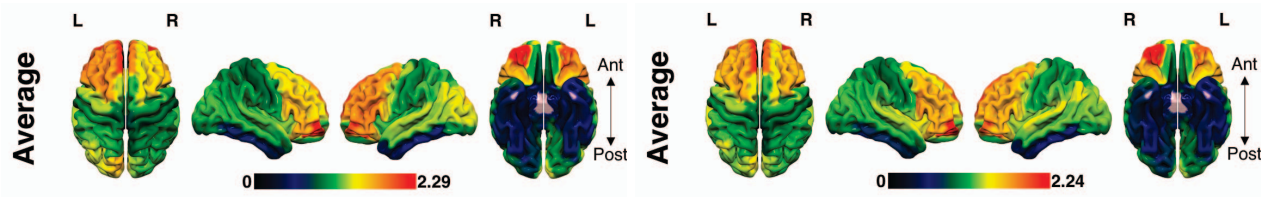
These results suggest that Code Review, Code Comprehension, and Prose Review all have largely distinct neural representations. Inspection of the average weight maps for each Code vs. Prose model (Figure 2) revealed a similar distribution of classifier weights across a number of brain regions (here, "hot" voxels push the decision function towards Code with greater activation, while "cool" voxels indicate the reverse). Correlating the voxelwise values confirmed a high degree of concordance ($r = 0.75$, $p < .001$), indicating that (on average) similar patterns of activity distinguished between code and prose regardless of which code task was being performed. In addition to the average classifiers, our highest-performing Code vs. Prose models also conserved the weight distributions across tasks. For space reasons we only present averages across all participants, showing the similarity in weight distributions. This general similarity helps explain why Code Review and Code Comprehension less separable than Code vs. Prose (indicated by lower classification accuracy).

B. RQ2 — Regional Inference

We investigate the relationship between tasks and particular brain regions. Compared to RQ1, which investigated whether classification was possible, RQ2 looks at the brain areas most involved in that classification and examines their traditional roles and importance.

Model	Class 1 Accuracy	Class 1 Z	Class 1 p	Class 2 Accuracy	Class 2 Z	Class 2 p	Overall BAC	Overall Z	Overall p
Review vs. Prose	58.33%	1.85	0.064	87.50%	4.20	2.63E-05	70.83%	4.00	6.34E-05
Comprehension vs. Prose	72.73%	2.74	0.006	95.83%	4.61	3.94E-06	79.17%	4.51	6.38E-06
Review vs. Comprehension	66.67%	3.68	2.32E-04	58.33%	0.90	0.366	61.84%	3.45	5.70E-04

TABLE I: Summary statistics for classifier performance. Median accuracies are given across participants; test statistics and probabilities are derived from nonparametric Wilcoxon signed-rank tests.



(a) Code Comprehension vs. Prose Review

(b) Code Review vs. Prose Review

Fig. 3: Average regional importance maps for task classifiers. "Hot" colors indicate areas containing a greater proportion of the total classification weight (over all 90 AAL-defined regions). These proportions are directly interpretable, such that extreme red regions are twice as "important" as light green regions. The left and right subfigures show a near-perfect correlation $r = 0.99$, $p < .001$, highlighting the same brain regions as important for both code tasks in general vs. the prose task.

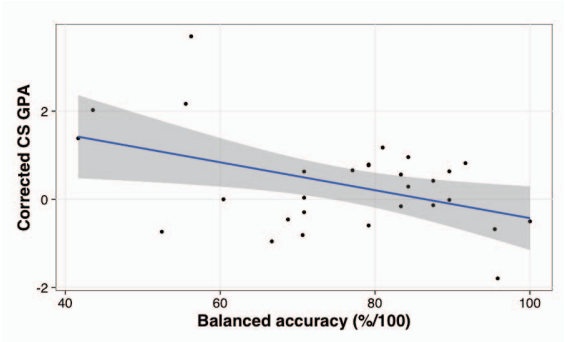


Fig. 4: Negative relationship between classifier performance (x-axis) and expertise (GPA), shaded 95% confidence interval.

As with the average multivariate weight maps, average regional importance maps for both Code vs. Prose classifiers demonstrated remarkable overlap (Figure 3). A correlation between importance maps yielded a near-perfect correspondence: $r = 0.99$, $p < .001$. For both classifiers, a wide swath of prefrontal regions known to be involved in higher-order cognition (executive control, decision-making, language, conflict monitoring, etc.) were highly weighted, indicating that activity in those areas strongly drove the distinction between code and prose processing. We also observed fairly large contributions from voxels near Wernicke's area in temporoparietal cortex — a region classically associated with language comprehension. Together, these results suggest that language-sensitive areas of the brain were differentially recruited when processing code versus English prose. Thus, on average, programming and natural languages exhibit unique neural representations.

C. RQ3 — Expertise

We examine the relationship between classifier accuracy and participant expertise.

In light of the observed variability in classification performance across individuals, coupled with stark differences in multivariate weight maps between the highest and lowest performing models (not shown), we tested whether *BAC* predicted one's programming expertise. As a proxy for expertise, we obtained undergraduate GPAs counting only courses from the Computer Science department. These were corrected by the total number of CS credits taken (a 4.0 GPA with 8 credits presumably does not indicate equal expertise to a 4.0 GPA with 32 credits): a simple linear regression was specified predicting computer science GPA from completed credits, and the residualized GPAs were extracted for subsequent analysis. This allowed us to consider GPA as a skill indicator *independent* of the number of credits completed.

We then computed the correlation between expertise and classifier accuracy for both of the Code vs. Prose models. Discriminability performance in Code Review vs. Prose models was not related to expertise ($r = -0.25$, $p = .184$). However, the extent to which classifiers distinguished between Code Comprehension and Prose significantly predicted expertise ($r = -0.44$, $p = .016$), see Figure 4. The inverse relationship between accuracy and expertise suggests that, as one develops more skill in coding, the neural representations of code and prose are less differentiable. That is, programming languages are treated more like natural languages with greater expertise.

D. Analysis Summary

We employed a data-driven machine learning approach to decode the neural representations of code and prose from multivariate patterns of brain activity. This technique is advantageous because it allows us to probe spatially-correlated

activations across the whole brain simultaneously. Binary Gaussian Process classifiers significantly predicted when a participant was performing code-related tasks relative to prose review; it also distinguished between the two code tasks, though to a lesser extent. This latter observation is consistent with the remarkable spatial overlap, both qualitatively and quantitatively, between multivariate classifier weights in Code vs. Prose models, suggesting that the code tasks were largely represented similarly on average. This was confirmed by nearly identical *a posteriori* estimates of regional importance: a number of prefrontal regions reliably distinguished between the code and prose tasks, accounting for most of the weight in the whole-brain classification models. Importantly, however, the extent to which these tasks were separable depended on one's expertise in programming — in the brains of experienced programmers, code and prose were nearly indistinguishable.

VI. Threats to Validity

Although our analyses show that neural representations of programming and natural languages are distinct (and this distinction reduces with expertise), this may not generalize.

One threat to validity associated with generalizability is that our code comprehension, code review, and prose review stimuli may not be indicative. For example, all code stimuli were in C and all prose stimuli were in English. While our examples are not multi-language we approach generality by choosing code changes at random from real-world projects and using established standardized test questions.

Another potential threat relates to construct and content validity: whether or not our tests measure what they claim to be measuring (e.g., "code comprehension" or "code review"). We mitigate this threat by posing types of questions known to be asked by programmers during software evolution tasks [71] and presenting code review as it would appear to a remote GitHub user. However, there are aspects of these tasks not covered by our current experiments (e.g., code review may also contain a back-and-forth conversational component).

Our use of GPA as a proxy for expertise introduces an additional threat to validity. Measuring participant expertise is difficult, and the metrics used are often domain-specific. For example, in specification mining the number of edits to version control repositories has been used as a proxy for expertise [44], while research related to Community Question Answering sites may proxy of expertise based on counting or profiling [83]. GPA has been shown to be a correlate of learning and academic aptitude (e.g., [28], [72]).

Finally, the high dimensionality of fMRI images and the complex mathematical methods required to analyze them often necessitate conservative corrections for false positives and/or strong assumptions about the underlying data (that may or may not be met by reality) [7]. For example, in standard GLM-based analyses of fMRI data, tens of thousands of statistical tests are run across the brain in a voxelwise fashion, requiring careful correction for multiple comparisons. In a highly-popularized article, Eklund *et al.* found that fMRI data often fail to meet the assumptions required for a certain "cluster-based" approach to multiple comparisons correction — this method, offered by nearly all common software packages for fMRI analysis, can therefore result in false-positive rates of up to 70% [23]. A key advantage to our multivariate approach is that all voxels are considered simultaneously, precluding the need for voxelwise multiple comparisons correction. However, this approach does preclude the sort of directed regional inference of standard GLM-based tests (cf. [70]).

VII. Costs and Reproducible Research

While we believe fMRI studies in software engineering are quite important, they remain rare (i.e., to the best of our knowledge this is only the second published instance [70]). In this section we frankly discuss the actual costs of carrying out this study in the hope that other researchers may carry out similar studies in the future. We note that of the four most common perceived barriers reported by software engineering researchers against the use of human studies [10, Fig. 10], fMRI experiments heavily involve all four: recruiting, experiment time, phrasing research questions, and the IRB.

a) Recruiting: fMRI constrains recruiting. Most directly, remote participation (such as via Amazon's Mechanical Turk crowdsourcing, cf. [26, Sec. 3.5]) is not possible. In addition, there are fMRI-specific filters, such as the exclusion of pregnant women, nearsighted people with glasses but not contacts, and the left-handed (because the location of language processing in the brain strongly depends on handedness). Despite this, we found recruiting to be quite easy. With brief advertisements in two CS classes, $100 reimbursements, and offering participants high-resolution scans of their brains (that can possibly be 3D-printed), we found 35 participants.

b) Time and Cost: Experiment time and cost are significant concerns for fMRI studies. One hour is about the maximum time that a participant can comfortably remain in the device. With pre- and post-screening, each participant thus takes about 90 minutes, and each participant must be separately supervised by one or two researchers (i.e., researchers listening to the participant via the intercom cannot leave the room during a scan). In addition, fMRI scan time is expensive — about $500 per hour at our institution. While both the fMRI costs and participant reimbursement can be paid for by an NSF grant with the appropriate budget justification and program manager permission, this is a significantly higher monetary cost than the usual software engineering human study. The data acquisition in this paper represents a participant and machine cost of $21,000 and 52.5 hours of graduate student time.

In addition, while most fMRI setups include a projector-and-mirror display, they may not be sufficient to show multiple lines of code clearly. We purchased an additional lens ($2800) to present indicative multi-line coding stimuli clearly.

c) Research Questions: The nature of the BOLD signal measured by fMRI influences experiment design. Notably, tasks in which participants are performing the same activities at the same time intervals are favored. Similarly, the contrasting, subtractive nature of fMRI analysis forces certain

experimental controls. Informally, fMRI cannot illuminate X directly: researcher must formulate tasks Y and Z such that X is the difference between them. In addition, the limited range of participant actions available restricts the range of tasks: for example, without a keyboard, no new coding is possible (but see Section VIII). In general, however, we found research question design to be fairly direct given consultations with psychology researchers who had fMRI experience.

d) IRB: An Institutional Review Board or Ethics Board governs acceptable human study research at a university. IRBs often distinguish between medical research and other (e.g., social or behavioral) research; fMRI studies fall into the heavily-regulated medical category. At the University of Virginia, the non-medical IRB protocol form involves six questions with 16 responses; those questions, plus potentially a few others involving compensation, comprise what is normally meant by (the burden of) IRB approval [9], [26], [27]. The medical IRB paperwork necessary for this study involved 236 questions in the cover sheet alone, and the main protocol was 33 pages (compared to 13 for non-medical [26]). In addition, since brain scans are HIPAA protected data, a four-page data protection and privacy plan was required.

e) Reproducible Research: To mitigate these costs, we have made our IRB protocol, experimental materials and raw, de-identified scan data publicly available at http://dijkstra.cs.virginia.edu/fmri/. This allows other researchers to conduct alternate analyses or produce more refined models without the expense of producing this raw data. There are instances of new papers published from archived human study data (e.g., [57] from [9]); we hope that the same will happen here.

f) Lessons Learned: One of the more difficult aspects of our experimental design was balancing ecological validity (i.e., are the "code review" and "code comprehension" activities undertaken by the subjects indicative of the real world) with the constraints of fMRI scanning. The available button press device precluded scrolling, requiring all stimuli to fit on one screen. The scan duration and data analysis precluded multi-minute tasks. We were also surprised by the number of mechanical difficulties in the fMRI apparatus that resulted in discarded participant runs (i.e., six out of 35). On a positive note, we were surprised by the number of willing students interested in obtaining a personal 3D-printed brain model.

VIII. Future Work

In this section we briefly describe three avenues for future research, both to give a sense of possible extensions and also to encourage collaboration from interested researchers.

First, one could adapt this protocol to study the impact of social relationships and patch provenance on code review. We propose to present patches as being written by, for example, a long-time co-worker, an offshore subcontractor, an automated tool, a recent hire, a deceptive adversary, or the experimenter in the room. Results in psychology suggest that perception of authority is enough to sway judgment in such cases [49]. Similarly, medical imaging research has shown that humans use different brain circuitry for social processing and cheating than for other types of reasoning [16], [75]. On the software engineering side, there is significant interest in the judgment of patch quality from various sources [26], [37], [74]. However, the extent and mechanism of mental biases in code review are not fully understood. This experiment also has the advantage that portions of it can be conducted without medical imaging.

Second, we are interested in a more thorough study of expertise. We envision collaboration with a research lab interested in mirroring our current protocol: one set of participants would be undergraduate and graduate university students, another would be industrial researchers with years of experience. Such a study might also explore code familiarity by drawing stimuli from codebases the experts or students had worked on. The small performance differences observed between students with two additional years of experience [27, Tab. 2] do not equal the order of magnitude reported in general [63, Tab. 3]. An imaging experiment involving actual expert programmers would allow us to determine in what way programming expertise changes the brain (cf. [17], [46], [65]).

Finally, our current experiments involve only reading code. This constraint is largely pragmatic: standard keyboards cannot be deployed near fMRI magnets. However, with minor cable modifications we were able to adapt a silicone-and-plastic keyboard [34] and obtain approval for its use in our fMRI. This would allow for the direct observation of patterns of brain activation while participants are *writing* code, both from scratch and to patch existing software.

IX. Summary

This paper presents the result of a controlled experiment in which code comprehension, code review, and prose review tasks are contrasted against each other using functional magnetic resonance imaging. Siegmund *et al.* asked whether, following Dijkstra, good programmers need good native language skills, and explored code but not language or expertise [70]. Hindle *et al.* found that most software admits the same statistical properties and modeling as natural language [31]. The work presented here in some sense bridges that gap, explicitly relating software, natural language and expertise.

We argue, at a high level, that medical imaging studies in computer science have the potential to shed light on multiple unresolved problems (e.g., unreliable self-reporting, pedagogy, retraining aging developers, technology transfer, expertise, and the relationship between software and natural language). We acknowledge that the work presented here is still quite exploratory: a full quantitative theory relating code, prose and expertise remains distant. We also acknowledge the time and material costs of such studies, and make our materials and data available, inviting collaboration on future work.

Empirically, we find that the neural representations of programming and natural languages are distinct. Our classifiers can distinguish between these tasks based solely on brain activity. We find that the same set of brain locations is relevant to distinguishing all three tasks. Finally, we find that expertise matters: greater skill accompanies a less-differentiated neural representation.

ACKNOWLEDGMENTS

Aspects of this work were partially supported by NSF grant CCF 1116289 and AFOSR grant FA8750-15-2-0075, as well as by Microsoft. We thank Jonathan Dorn, Jeremy Lacomis, and Kevin Angstadt for helpful suggestions on earlier drafts.

REFERENCES

[1] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski. Software inspections: An effective verification process. *IEEE Softw.*, 6(3):31–36, May 1989.

[2] A. Arcuri and L. Briand. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability (STVR)*, 24(3):219–250, 2014.

[3] M. Atherton, J. Zhuang, W. M. Bart, X. Hu, and S. He. A functional MRI study of high-level cognition. I. the game of chess. *Cognitive Brain Research*, 16(1):26–31, 2003.

[4] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou. Evaluating static analysis defect warnings on production software. In *Program Analysis for Software Tools and Engineering*, pages 1–8, 2007.

[5] T. Ball, B. Cook, V. Levin, and S. K. Rajamani. SLAM and static driver verifier: Technology transfer of formal methods inside microsoft. In *Integrated Formal Methods*, pages 1–20, 2004.

[6] D. S. Bassett, N. F. Wymbs, M. A. Porter, P. J. Mucha, J. M. Carlson, and S. T. Grafton. Dynamic reconfiguration of human brain networks during learning. *Proceedings of the National Academy of Sciences*, 108(18):7641–7646, 2011.

[7] C. M. Bennett, M. Miller, and G. L. Wolford. Neural correlates of interspecies perspective taking in the post-mortem atlantic salmon: An argument for proper multiple comparisons correction. *NeuroImage*, 47, July 2009.

[8] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. R. Engler. A few billion lines of code later: Using static analysis to find bugs in the real world. *Communications of the ACM*, 53(2):66–75, 2010.

[9] R. P. Buse and W. Weimer. Learning a metric for code readability. *IEEE Trans. Software Eng.*, Nov. 2009.

[10] R. P. L. Buse, C. Sadowski, and W. Weimer. Benefits and barriers of user evaluation in software engineering research. In *Object-Oriented Programming, Systems, Languages and Applications*, pages 643–656, 2011.

[11] R. P. L. Buse and T. Zimmermann. Information needs for software development analytics. In *International Conference on Software Engineering*, pages 987–996, 2012.

[12] R. Cabeza, S. M. Daselaar, F. Dolcos, S. E. Prince, M. Budde, and L. Nyberg. Task-independent and task-specific age effects on brain activity during working memory, visual attention and episodic retrieval. *Cerebral Cortex*, 14(4):364–375, 2004.

[13] L. F. Capretz, D. Varona, and A. Raza. Influence of personality types in software tasks choices. *Computers in Human Behavior*, 52:373–378, 2015.

[14] E. Challis, P. Hurley, L. Serra, M. Bozzali, S. Oliver, and M. Cercignani. Gaussian process classification of Alzheimer's disease and mild cognitive impairment from resting-state fMRI. *Neuroimage*, 122:232–243, 2015.

[15] M. T. H. Chi, R. Glaser, and E. Rees. Expertise in problem solving. *Advances in the psychology of human intelligence*, 1:7–76, 1982.

[16] L. Cosmides and J. Tooby. Can a general deontic logic capture the facts of human moral reasoning? How the mind interprets social exchange rules and detects cheaters. *Moral psychology*, pages 53–119, 2008.

[17] U. Debarnot, M. Sperduti, F. Di Rienzo, and A. Guillot. Experts bodies, experts minds: How physical and mental training shape the brain. *Frontiers in Human Neuroscience*, 8:280, 2014.

[18] L. E. Deimel Jr. The uses of program reading. *SIGCSE Bulletin*, 17(2):5–14, 1985.

[19] J. Diedrichsen and R. Shadmehr. Detecting and adjusting for artifacts in fmri time series data. *NeuroImage*, 27(3):624–634, 2005.

[20] W. J. Doherty and A. J. Thadani. The economic value of rapid response time. In *IBM Systems Journal*, Nov. 1982.

[21] A. Dunsmore, M. Roper, and M. Wood. Practical code inspection techniques for object-oriented systems: An experimental comparison. *IEEE Softw.*, 20(4):21–29, July 2003.

[22] L. Dupre. *BUGS in Writing: A Guide to Debugging your Prose*. Addison-Wesley Professional, second edition, 1998.

[23] A. Eklund, T. E. Nichols, and H. Knutsson. Cluster failure: Why fmri inferences for spatial extent have inflated false-positive rates. *Proceedings of the National Academy of Sciences*, 113(28):7900–7905, 2016.

[24] J. L. Elshoff and M. Marcotty. Improving computer program readability to aid modification. *Commun. ACM*, 25(8):512–521, 1982.

[25] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Syst. J.*, 38(2-3):258–287, June 1999.

[26] Z. P. Fry, B. Landau, and W. Weimer. A human study of patch maintainability. In *International Symposium on Software Testing and Analysis*, pages 177–187, 2012.

[27] Z. P. Fry and W. Weimer. A human study of fault localization accuracy. In *International Conference on Software Maintenance*, pages 1–10, 2010.

[28] W. A. Grove, T. Wasserman, and A. Grodner. Choosing a proxy for academic aptitude. *Journal of Economic Education*, pages 131–147, 2006.

[29] P. Hallam. What do programmers really do anyway? Technical report, Microsoft Developer Network, 2016.

[30] N. J. Haneef. Software documentation and readability: a proposed process improvement. *SIGSOFT Softw. Eng. Notes*, 23(3):75–77, 1998.

[31] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *International Conference on Software Engineering*, pages 837–847, 2012.

[32] D. Huizinga and A. Kolawa. *Automated Defect Prevention: Best Practices in Software Management*. Wiley, first edition, 2007.

[33] Y. Ikutani and H. Uwano. Brain activity measurement during program comprehension with NIRS. In *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 1–6, 2014.

[34] G. A. James, G. He, and Y. Liu. A full-size MRI-compatible keyboard response system. *Neuroimage*, 25(1):328–31, Mar. 2005.

[35] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *International Conference on Software Engineering (ICSE)*, pages 672–681. IEEE, 2013.

[36] N. Kennedy. Google Mondrian: web-based code review and storage. In *http://www.niallkennedy.com/blog/2006/11/google-mondrian.html*, Nov. 2006.

[37] D. Kim, J. Nam, J. Song, and S. Kim. Automatic patch generation learned from human-written patches. In *International Conference on Sofware Engineering*, 2013.

[38] J. C. Knight and E. A. Myers. An improved inspection technique. *Commun. ACM*, 36(11):51–61, Nov. 1993.

[39] D. E. Knuth. Literate programming. *Comput. J.*, 27(2):97–111, 1984.

[40] M. V. Kosti, R. Feldt, and L. Angelis. Archetypal personalities of software engineers and their work preferences: a new perspective for empirical studies. *Empirical Software Engineering*, 21(4):1509–1532, 2016.

[41] M. Kuss and C. E. Rasmussen. Assessing approximate inference for binary Gaussian process classification. *Journal of Machine Learning Research*, 6:1679–1704, 2005.

[42] Laboratory of Neuro Imaging. Human connectome project. In *http://www.humanconnectomeproject.org/, Martinos Center for Biomedical Imaging at Massachusetts General Hospital*, Aug. 2016.

[43] S. LaConte, S. Strother, V. Cherkassky, and X. Hu. Support vector machines for temporal classification of block design fmri data. *Neuroimage*, 26:317–329, 2005.

[44] C. Le Goues and W. Weimer. Measuring code quality to improve specification mining. *IEEE Transactions on Software Engineering*, 38(1):175–190, 2012.

[45] P. Mabe and S. West. Validity of self-evaluation of ability: A review and meta-analysis. *Journal of Applied Psychology*, 67(3):280–296, 6 1982.

[46] E. A. Maguire, K. Woollett, and H. J. Spiers. London taxi drivers and bus drivers: A structural mri and neuropsychological analysis. *Hippocampus*, 16(12):1091–1101, 2006.

[47] J. B. A. Maintz and M. A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.

[48] A. Marquand, M. Howard, M. Brammer, C. Chu, S. Coen, and J. Mourao-Miranda. Quantitative prediction of subjective pain intensity from whole-brain fMRI data using Gaussian processes. *Neuroimage*, 49:2178–2189, 2010.

[49] S. Milgram. Behavioral study of obedience. *Abnormal and Social Psychology*, 67(4):371–378, Oct. 1963.

[50] M. P. Milham, K. I. Erickson, M. T. Banich, A. F. Kramer, A. Webb, T. Wszalek, and N. J. Cohen. Attentional control in the aging brain: Insights from an fMRI study of the Stroop task. *Brain and Cognition*, 49(3):277–296, 2002.

[51] J. A. Mumford, B. O. Turner, F. G. Ashby, and R. A. Poldrack. Deconvolving BOLD activation in event-related designs for multivoxel pattern classification analyses. *NeuroImage*, 59(3):2636–2643, 2012.

[52] 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 *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 448–451, New York, NY, USA, 2014. ACM.

[53] NASA Software Reuse Working Group. Software reuse survey. In *http://www.esdswg.com/softwarereuse/Resources/library/working_group_doc%uments/survey2005*, 2005.

[54] S. Ogawa, T. M. Lee, A. R. Kay, and D. W. Tank. Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of the National Academy of Sciences*, 87(24):9868–9872, 1990.

[55] C. Parnin and A. Orso. Are automated debugging techniques actually helping programmers? In *International Symposium on Software Testing and Analysis*, pages 199–209, 2011.

[56] P. M. Podsakoff and D. W. Organ. Self-reports in organizational research: Problems and prospects. *Journal of Management*, 12(4):531–544, 1986.

[57] D. Posnett, A. Hindle, and P. T. Devanbu. A simpler model of software readability. In *Mining Software Repositories*, pages 73–82, 2011.

[58] C. E. Rasmussen and C. Williams. *Gaussian Processing for Machine Learning*. MIT Press, 2006.

[59] D. R. Raymond. Reading source code. In *Conference of the Centre for Advanced Studies on Collaborative Research*, pages 3–16, 1991.

[60] M. Ritchey, A. P. Yonelinas, and C. Ranganath. Functional connectivity relationships predict similarities in task activation and pattern information during associative memory encoding. *J. Cognitive Neuroscience*, 26(5):1085–1099, May 2014.

[61] J. S. Ross, J. Tkach, P. M. Ruggieri, M. Lieber, and E. Lapresto. The minds eye: Functional MR imaging evaluation of golf motor imagery. *American Journal of Neuroradiology*, 24(6):1036–1044, 2003.

[62] S. Rugaber. The use of domain knowledge in program understanding. *Ann. Softw. Eng.*, 9(1-4):143–192, 2000.

[63] H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing online and offline programming performance. *Commun. ACM*, 11(1):3–11, Jan. 1968.

[64] B. Schölkopf and A. J. Smola. *Learning with kernels*. The MIT Press, 2002.

[65] J. Scholz, M. C. Klein, T. E. J. Behrens, and H. Johansen-Berg. Training induces changes in white-matter architecture. *Nature Neuroscience*, 12:1370–1371, 2009.

[66] J. Schrouff, J. Cremers, G. Garraux, L. Baldassarre, J. Mourao-Miranda, and C. Phillips. Localizing and comparing weight maps generated from linear kernel machine learning models. In *International Workshop on Pattern Recognition in Neuroimaging (PRNI)*, pages 124–127, 2013.

[67] J. Schrouff, C. Kusse, L. Wehenkel, P. Maquet, and C. Phillips. Decoding semi constrained brain activity from fMRI using support vector machines and Gaussian processes. *PLOS One*, 7(4):1–11, 2012.

[68] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

[69] F. Shull, I. Rus, and V. Basili. Improving software inspections by using reading techniques. In *International Conference on Software Engineering*, pages 726–727, 2001.

[70] 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 *International Conference on Software Engineering*, pages 378–389, 2014.

[71] J. Sillito, G. C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Foundations of Software Engineering*, pages 23–34, 2006.

[72] A. Solimeno, M. E. Mebane, M. Tomai, and D. Francescato. The influence of students and teachers characteristics on the efficacy of face-to-face and computer supported collaborative learning. *Computers & Education*, 51(1):109–128, 2008.

[73] I. Sommerville. *Software Engineering*. Pearson, ninth edition, 2010.

[74] M. Soto, F. Thung, C.-P. Wong, C. Le Goues, and D. Lo. A deeper look into bug fixes: Patterns, replacements, deletions, and additions. In *Mining Software Repositories*, pages 512–515, 2016.

[75] V. E. Stone, L. Cosmides, J. Tooby, N. Kroll, and R. T. Knight. Selective impairment of reasoning about social exchange in a patient with bilateral limbic system damage. *Proceedings of the National Academy of Sciences of the United States of America*, 99(17):11531–6, 2002.

[76] The College Board. *The Official SAT Study Guide (Redesigned SAT)*. 2016.

[77] A. Tsotsis. Meet Phabricator, the witty code review tool built inside Facebook. In *https://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath-penned-thes%e-words/*, Aug. 2011.

[78] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot. Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *NeuroImage*, 15(1):273–289, 2002.

[79] C. Weiß, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Workshop on Mining Software Repositories*, May 2007.

[80] Wellcome Trust Centre for Neuroimaging. Statistical parametric mapping. In *http://www.fil.ion.ucl.ac.uk/spm/*, Aug. 2016.

[81] L. Williamson. IBM Rational software analyzer: Beyond source code. In *Rational Software Developer Conference*, June 2008.

[82] J. Wright. In-demand and aging: A look at engineers and engineering technicians in the workforce. Technical Report http://www.economicmodeling.com/2014/09/12/in-demand-and-aging-a-look-at-engineers-and-engineering-technicians-in-the-workforce/, EMSI, 2014.

[83] R. Yeniterzi and J. Callan. Moving from static to dynamic modeling of expertise for question routing in CQA sites. In *International Conference on Web and Social Media*, pages 702–705, 2015.

[84] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. N. Bairavasundaram. How do fixes become bugs? In *Foundations of Software Engineering*, pages 26–36, 2011.

[85] M. B. Zanjani, H. H. Kagdi, and C. Bird. Automatically recommending peer reviewers in modern code review. *IEEE Trans. Software Eng.*, 42(6):530–543, 2016.