Using (Bio)Metrics to Predict Code Quality Online

Sebastian C. Müller, Thomas Fritz
Department of Informatics, University of Zurich, Switzerland
{smueller, fritz}@ifi.uzh.ch

ABSTRACT

Finding and fixing code quality concerns, such as defects or poor understandability of code, decreases software development and evolution costs. A common industrial practice to identify code quality concerns early on are code reviews. While code reviews help to identify problems early on, they also impose costs on development and only take place after a code change is already completed. The goal of our research is to automatically identify code quality concerns while a developer is making a change to the code. By using biometrics, such as heart rate variability, we aim to determine the difficulty a developer experiences working on a part of the code as well as identify and help to fix code quality concerns before they are even committed to the repository.

In a field study with ten professional developers over a two-week period we investigated the use of biometrics to determine code quality concerns. Our results show that biometrics are indeed able to predict quality concerns of parts of the code while a developer is working on, improving upon a naive classifier by more than 26% and outperforming classifiers based on more traditional metrics. In a second study with five professional developers from a different country and company, we found evidence that some of our findings from our initial study can be replicated. Overall, the results from the presented studies suggest that biometrics have the potential to predict code quality concerns online and thus lower development and evolution costs.

1. INTRODUCTION

A commonly accepted principle in software evolution is that delaying software quality concerns, such as defects or poor understandability of the code, increases the cost of fixing them [10,11,41,45]. Ward Cunningham even went as far as stating that *"every minute spent on not-quite-right code counts as interest on that debt"* [20].

Code reviews are one practice that is widely used today to detect code quality problems early on. Code reviews are generally performed by peers after a developer completes the changes for a task and they help to improve code, *e.g.* its readability, and to find defects [6, 12, 31, 62]. At the same time, code reviews impose costs in terms of time and effort by peers to perform the review. Several automatic approaches to detect code quality concerns have been proposed, for instance, to detect defects [51, 52, 76] or code smells [39, 70]. These approaches generally have two disadvantages in common: first, they are predominantly based on metrics, such as code churn or module size, that can only be collected after a code change is completed and often require access to further information, such as the history of the code; second, they do not take the individual differences between developers comprehending code into account, such as the ones that exist between novices and experts [19].

The goal of our work is to use biometric sensing to overcome these disadvantages and lower the development cost by identifying code quality concerns online—while a developer is still working on the code. Previous research, including one of our earlier studies, has already shown that certain biometric measures, such as heart rate variability (HRV) or electrodermal activity (EDA), can be linked to task difficulty or difficulty in comprehending small code snippets [29, 53, 71, 72]. The general concepts behind these studies are that biometric measures can be used to determine cognitive load—the amount of mental effort required to perform a task—and that the more difficult a task, the higher the cognitive load, and the higher the error rate [5,68]. In our work, we build on top of these concepts and aim to examine the use of biometrics to determine the places in the code that professional developers perceive to be difficult and are therefore more likely to contain quality concerns (errors). This would allow us to automatically perform preemptive code reviews helping developers to commit code with less quality concerns.

To investigate the use of biometrics to predict code quality concerns online, we performed a field study with ten professional developers in a Canadian company over a period of two weeks. We collected a variety of metrics, including biometrics, such as heart rate variability, as well as more traditional metrics, such as code complexity and churn. After each committed change and periodically throughout the study, we asked developers to assess the perceived difficulty of the code elements—methods and classes—they were just working with. Additionally, we collected quality concerns identified in peer code reviews of the committed changes. Amongst other results, our study shows that biometrics outperform more traditional metrics and a naive classifier in predicting a developer's perceived difficulty of code elements while working on these elements. Our analysis also shows that code elements that are perceived more difficult by de-

ICSE '16, May 14 - 22, 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3900-1/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2884781.2884803

velopers also end up having more quality concerns found in peer code reviews, which supports our initial assumption. In addition, the results show that biometrics helped to automatically detect 50% of the bugs found in code reviews and outperformed traditional metrics in predicting all quality concerns found in code reviews.

To assess our approach's generalizability, we conducted a second study with five developers in a Swiss company over a period of a week. The results of this study provide evidence that some (but not all) of our findings can be replicated.

In summary, this paper makes the following contributions:

- It presents results of a two-week study with ten developers investigating the use of biometrics in the field to determine code quality concerns and developers' perceived difficulty.
- It provides a comparison between various metrics, showing that biometrics can outperform more traditional metrics in predicting code quality concerns online.
- It presents results of a one-week replication study with five developers from a different company and country.

Overall, the results of our studies suggest that developers' biometrics have potential to identify difficult places in the code and in turn quality concerns and thus might be used to lower the overall software maintenance cost.

2. RELATED WORK

Work related to our research can roughly be categorized into three major areas: the manual detection of quality concerns in form of inspections and code reviews, automatic detection based on code, change and interaction metrics, and, more broadly, the use of biometrics in software development.

Manual Detection. The substantial benefits and cost savings of manual software inspection have long been known based on evidence from multiple places [1, 22, 32]. While these results were mostly based on formal inspections, companies today often employ more lightweight and tool supported code review processes that require less time and effort [25]. Several studies have looked into these lighter weight code reviews, in particular their practices, characteristics and outcomes, and shown amongst other results, that these lightweight code reviews still lead to substantial code improvements and the detection of defects [6, 12, 62]. Overall the results from these studies show that manual code inspection can help to detect many quality concerns soon after code changes were performed and lead to significant cost savings in software evolution. At the same time, manual inspections still require time and effort of peer developers and can only be done after the code was committed or shared for review.

Automatic Detection. There is a myriad of research investigating the automatic detection of code quality concerns. Most of these approaches focus on various software metrics, such as complexity, size or change metrics and their correlation to software defects [50, 51, 76]. Instead of code and change metrics, researchers have also looked into organizational information to predict defects, for instance the number of developers who touched a file [52, 74]. Others have focused on the automatic detection of code smells, predominantly by using code metrics in combination with absolute or relative thresholds or rule sets [2, 44, 46, 49], but also by mining the change history [55]. Furthermore, there are tools, such as FindBugs or PMD that can help to identify potential quality concerns in the code [28, 57]. Most of these approaches only allow for a post-hoc classification and do not take into account the individual differences between developers working on the code. A first step into this direction was taken by Lee *et al.* [40] who focused on developers' individual interaction patterns and proposed 56 micro interaction metrics for defect prediction. In a case study, the authors compared defect prediction learners based on change metrics and source code metrics with models based on micro interaction metrics, and found that developers' interaction patterns, such as the ratio between edits and selects, can significantly improve the defect prediction.

Rather than identifying quality concerns, code metrics have also been used to assess the difficulty of various code-related activities, such as program comprehension. For instance, Curtis *et al.* [21], or Feigenspan *et al.* [27], investigated how different kinds of code metrics correlate with developers' performance on maintenance tasks, respectively program comprehension. Closer to our research, Carter *et al.* [15] used interaction logs within the IDE to predict when a developer is stuck, experiencing a lot of difficulties and cannot make any progress.

In contrast to these studies, we are investigating the use of biometric measurements to identify quality concerns. Using biometrics would allow for an online detection that takes into account the individual differences between developers.

Biometrics. In psychology, a broad range of biometric measurements has been investigated and correlated with a person's cognitive states and processes. These biometrics can be roughly categorized into skin-, heart- and breathing-related measurements. Commonly used measurements are electrodermal activity (EDA) and skin temperature for the skin-, heart rate (HR) and heart rate variability (HRV) for the heart- and the respiratory rate (RR) for breathing-related measurements. For all these measurements, researchers have found correlations to mental and cognitive load/effort, as well as to task difficulty as presented in Table 1.

In the context of software engineering, these biometric measurements were used to assess developers' mental load and perceived difficulty while working on small code snippets. Parnin [56] investigated the potential of electromyography (EMG) to measure sub-vocal utterances and found that this might be used to determine programming task difficulty. In a similar direction, Nakagawa *et al.* [53] used Near Infrared Spectroscopy (NIRS) to measure developers' cerebral blood flow (CBF) while working on code comprehension tasks with two difficulty levels. Radevski *et al.* [59] proposed an approach that uses electro-encephalography (EEG) to assess developers' productivity in real time. Finally, in a previous study, we used a combination of biometric sensors and found that they can be used to predict the difficulty of small code comprehension tasks [29]. Besides these studies, most research in software engineering using biometric measurements focused on eye tracking technology (*e.g.* [7, 19, 63]) or developers' brain activities (*e.g.* [34, 66]) to gain a better understanding of program comprehension. In contrast to all these studies, we focus on the online prediction of code quality concerns and are one of the first ones to perform a longitudinal two-week field study with biometrics sensors.

3. PSYCHOLOGICAL BACKGROUND

Our work on the use of biometrics builds on top of established psychological research and concepts, including the

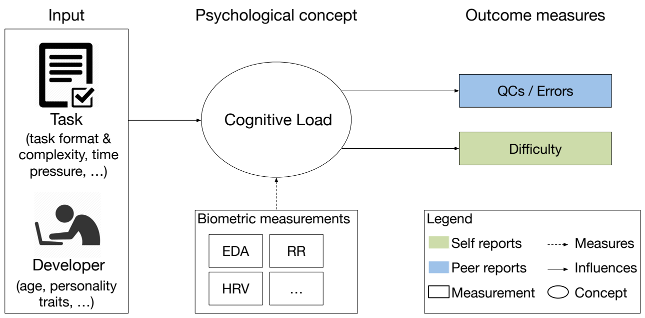


Figure 1: Concepts and relationships between input, cognitive load, biometric measurements, and outcome.

cognitive load theory. Figure 1 illustrates some of these relations relevant to our work. Cognitive load (CL) refers to the required mental effort to perform a task and is composed of intrinsic (*e.g.* inherent task difficulty), extrinsic (*e.g.* the way the code is written) and germane load (effort for processing information) [69]. In general, the more difficult it is to perform a task for an individual, the higher the cognitive load, and in turn, the lower the individual's performance and the higher the error rate [5,36,69,73]. Previous studies have shown that mental effort and cognitive load can be measured using biometrics (*e.g.* [71, 75]). Based on the links between cognitive load, errors and biometrics, we might be able to use biometric measurements to determine a developer's perceived difficulty when working with a code element and the likelihood of an error being created. Also, since biometrics are linked directly to a developer's cognitive load and thus capture individual differences even for the same task, this approach should be more accurate than proxies that try to capture cognitive load from artifacts.

4. STUDY METHOD

To investigate the use of biometrics to detect quality concerns online, we analyzed four main research questions:

RQ1 Can we use biometrics to identify places in the code that are perceived to be more difficult by developers?

RQ2 Can we use biometrics to identify code quality concerns found through peer code reviews?

RQ3 How do biometrics compare to more traditional metrics for detecting quality concerns?

RQ4 How sensitive are these biometrics to the individual?

To address our research questions, we conducted a long-term empirical field study with ten professional software developers, working for a medium-sized software development company in Canada. Over the course of two weeks participants worked on their usual tasks in their usual work environment while wearing biometric sensors. We periodically asked participants to rate the difficulty of the code elements—methods and classes—they were working with on a 6-point Likert scale and collected quality concerns identified in peer code reviews. In addition, we gathered more traditional metrics for comparison purposes[1].

[1]A replication package of this study is available online [60].

4.1 Participants & Sensors

We were able to recruit ten professional software developers from a medium-sized software development company in Canada for our study. The ten participants (nine male, one female) ranged in age from 23 to 45 years and had an average professional software engineering experience of 10.2 years (±6.2), ranging from 3 to 22 years. All study participants worked on the same project, but were split over three different teams that were in charge of different components of the project. All teams followed a similar agile software development process and worked on tasks with similar sizes[2]. Each participant had access to her biometric data and was allowed to quit any time without providing a reason.

We used two biometric sensors for this study: an Empatica E4 wristband [24] to capture skin- and heart-related measurements, and a SenseCore chest strap[3] to capture skin- (except for EDA), heart-, and breathing-related measurements. Participants were asked to wear the chest strap and optionally also the wristband. We ended up with all ten participants wearing the SenseCore sensor for the two-week study period, and six of them (P01, P04, P05, P06, P07 and P08) also wearing the Empatica wristband in addition.

4.2 Study Procedure

At the beginning of the study, we asked each participant to install a small, self-written interaction monitor plugin into their Eclipse IDE that logged each time a developer selected or edited a method or a class within the IDE in combination with the current timestamp. At the same time, the plugin collected a set of code metrics for each code element that was selected or edited. After that, we introduced participants to the biometric sensors, gave them the option of either wearing both or just the SenseCore chest strap, and helped them to put the sensor(s) on for the first time. Then we told participants to continue performing their work as usual for the next two weeks while wearing the biometric sensors. An overview of the procedure and data collection for the two-week period is presented in Figure 2.

At the end of each workday, we collected the biometric data from each participant and charged the batteries of the sensor(s). Once per day of the study, participants were also asked to watch a two minutes video of fish swimming in a fish tank while wearing the biometric sensor(s). The movie was intended to help participants relax and allow us to record a baseline during the second minute of the two minutes session that we used later on to normalize the captured biometric data. In previous studies [29, 48] we saw that a person's biometric features drop back to a baseline after about a minute of watching the video.

In addition to the code metrics and interaction logs that we recorded with our Eclipse plugin, we also collected three different types of outcome measures. First, every 90 minutes, the Eclipse plugin prompted participants to answer a small questionnaire within Eclipse that asked them to rate the perceived difficulty of 20 randomly selected code elements that they were working with within the last 90 minutes. Second, every time a participant committed a set of code changes to the repository, we asked the participant

[2]For privacy reasons we are not able to disclose more specifics on the company and also substituted code element names throughout the paper.

[3]SenseCore sensors are no longer available due to the company's closure.

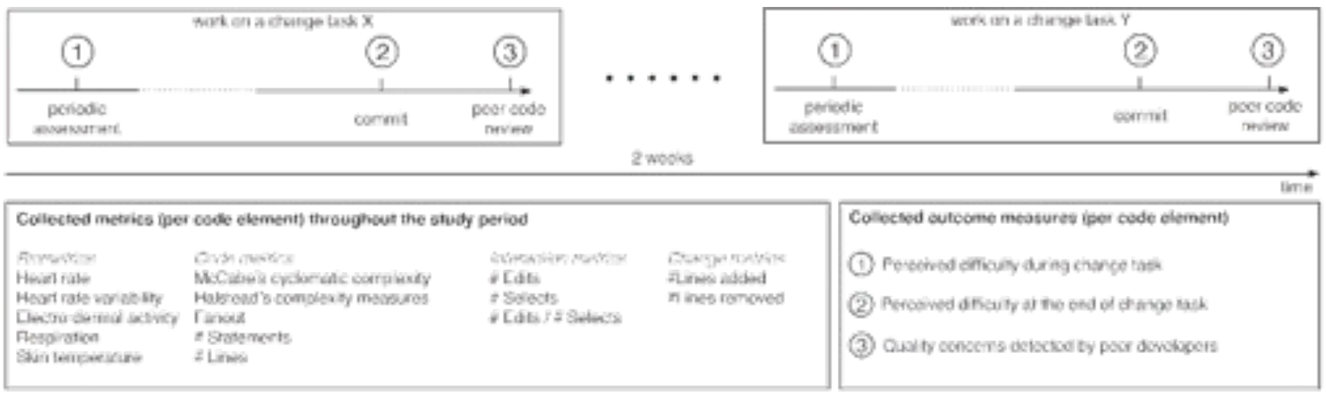


Figure 2: Overview of study procedure and collected data.

Table 1: Overview of collected biometrics and their previously found correlations to psychological aspects.

Measurement	Previously found correlations
Heart-related	
Heart rate variability	mental effort [71]; task difficulty [72]; mental load [61, 75]; task demand [26]
Heart rate	mental effort [71]; mental load [61, 75]; task difficulty [3, 18, 72]
Breathing-related	
Respiratory rate	mental effort [71]; task difficulty [37]; task demand [26]
Skin-related	
Skin temperature	task difficulty [3, 18, 54]
Electro-dermal activity	mental load [61, 75]; task difficulty [18, 54]; stress and cognitive load [65]

to rate the difficulty s/he perceived while working on and changing each of the classes and methods in the committed changes. As part of the company's development process, each committed code change was also reviewed by one to three peers. Finally, we collected the outcome of the code reviews for the code changes committed by our participants.

At the end of the study, each participant completed a short demographic background questionnaire.

4.3 Metrics and Outcome Measures

We collected four kinds of metrics—biometrics, code metrics, interaction metrics, and change metrics—and three different types of outcome measures.

4.3.1 Biometrics

We used the chest strap and the wristband to collect various biometric measurements that have previously been linked to task difficulty as well as cognitive and mental effort. Table 1 provides an overview of the biometrics we captured for this study and the previously found correlations. A complete list of all extracted features can be found in the replication package [60].

To use biometric data for predicting quality concerns of code elements, we had to apply several data segmentation, data cleaning and feature extraction steps. The biometric sensor data is captured as a sequence of data entries with a timestamp and the person's biometric values for that point in time. To map biometric data to code elements, we used the assumption that a developer is thinking about and affected by the code element s/he just selected or edited (see Section 8) and therefore segmented the biometric data based on the user interaction log that we captured with our Eclipse plugin. Specifically, we used the time interval from the point in time a developer interacted with a code element C in the IDE up to the point in time s/he interacted with a different element or left the IDE to segment the biometric data and associated the biometric data segment corresponding to this time interval with the code element C. Since a person's heart rate and the phasic part of the EDA signal typically take about one to two seconds to adapt to changes [9,64,67], we only considered segments that span at least three seconds in our analysis, *i.e.* when the developer spent at least three seconds on a code element before moving on, and filtered the biometric data for the first two seconds of the segment to allow for the change in the biometrics to take place.

Heart-related biometrics. For the heart rate, we extracted the mean and the variance of the signal, while for the heart rate variability, we used features that represent the difference in time between two heart beats, such as RMSSD (root mean square of successive differences) or NN50 (the number of pairs of successive beat-to-beat intervals that differ more than 50ms). All these features have been linked to mental effort and load as well as task difficulty [3, 71].

Breathing-related biometrics. Previous research linked a person's respiratory rate to task difficulty [26,37]. We therefore extracted commonly used features, such as the mean respiration rate or the $\log_{10}$ variance of the respiration signal and added them to our feature set.

Skin-related biometrics. For the skin temperature, we extracted features, such as the mean temperature that research has linked to task difficulty [3, 18]. To extract features from the EDA signal, we used Butterworth filters [14] to split it into two parts: the high frequency, fast changing phasic part, and the low frequency, slowly changing tonic part [64]. In a next step, we extracted features related to the peaks in the phasic signal, and features from the tonic part that research has linked to mental load and task difficulty [54, 61, 75].

All biometric measurements were normalized using the baseline measurements that we collected during the second minute of the two minutes fish tank movie.

4.3.2 Code, Change and Interaction Metrics

We collected several metrics for code elements—methods and classes—that have previously been associated with difficult code or defects. Most of these metrics were captured with our Eclipse plugin.

Code metrics. For each code element, the plugin calculated code metrics that research has linked to difficulty in program comprehension and code quality. The collected metrics were McCabe's complexity (*e.g.* [51]), Halstead's complexity measures (*e.g.* [23]), various size metrics (*e.g.* [17]), and fanout (*e.g.* [77]). Since code metrics might alter when a developer makes changes to a code element, we captured the metrics every time a developer interacted with a code element.

Change metrics. Every time a developer committed a change set to the repository in the study period, we extracted the number of lines added and removed for each code element. Previous research has shown that these metrics can be reliable predictors for defects (*e.g.* [47,50]). Due to limited access to the source code repositories in the company, we were only able to collect these metrics on class, and not on method level.

Interaction metrics. Previous research has shown that metrics on interaction data, in particular the ratio between edit and select events, might be used to improve defect prediction and to determine when a developer experiences difficulties (*e.g.* [15, 40]). We therefore collected the number and ratio of edit and select events for each code element.

4.3.3 Outcome Measures

Over the course of the study, we collected three different types of outcome measures.

Perceived difficulty during a change task. Every 90 minutes, participants were prompted with a questionnaire that asked them to rate the difficulty they perceived while working on 20 randomly selected code elements from the previous 90 minutes on a 6-point Likert scale (from 1 = "very easy" to 6 = "very difficult"). For the 20 elements in each questionnaire, we equally balanced the number of methods and classes and the number of edited and selected code elements, unless the participant did not interact with a sufficient number of elements in the previous 90 minutes.

Perceived difficulty at the end of a change task. We manually monitored code repositories. As soon as we noticed that a developer committed a change set to the repository, we asked her/him to rate the difficulty s/he perceived while performing the necessary changes for each class and method that was changed. For this rating, we used the same 6-point Likert scale as for the first outcome measure.

Code quality concerns detected through peer reviews. Each committed change set was typically reviewed by one to three peers shortly after the commit time. The reviewers looked for actual bugs, inadequate documentation or test cases, and violations of coding styles. We collected the results of these code reviews for each change set that was committed by one of the study participants. We marked a code element as containing a quality concern when at least one was identified in a code review.

4.4 Data Collection

Across all study participants and the two weeks of the study, we were able to collect biometric measurements for a total of 116 developer work days (∅=11.6, ±1.8). This resulted in 12.1 GB worth of biometric data, consisting of 40.6 million data points. For all ten study participants, we collected skin temperature, HR, HRV and RR data. For six study participants who volunteered to also wear the Empatica wristband sensor, we were able to collect the EDA as well as a second skin temperature and HR(V) measurement. We decided to take the signal from the SenseCore sensor whenever possible and only rely on the Empatica signal in case the SenseCore signal could not be recorded, since our previous experiences with the two sensors indicate that the SenseCore signal is more accurate.

In addition to the biometric data, we collected perceived difficulty ratings for 1511 methods and 1480 classes. From the 1511 difficulty ratings for methods, 982 were collected while developers were working on a change task, while the rest were collected at commit time. Similarly, 900 classes were rated while working on a change task and 580 at commit time. On average, study participants spent 12.0 minutes on a particular class and 6.8 minutes on a particular method, between two consecutive difficulty ratings that occurred every 90 minutes. Table 2 provides an overview of the difficulty ratings we collected for each participant in the study. For all code elements that were changed and committed by one of our participants, we were also able to collect the results of the peer code reviews of these elements. In total, we collected 162 quality concerns, 46 on method level and 116 on class level. We ended up with 95 (16.4%) classes in which a quality concern was found and 485 (83.6%) without any quality concern. Similarly, our dataset consists of 44 (8.3%) methods with a quality concern and 485 (91.7%) methods without any quality concern. Table 3 provides an overview of the categories of quality concerns found during code reviews. Finally, we also collected answers to the demographic questionnaires at the end of the study.

Table 2: Number of collected data points for each study participant during and at the end of a change task.

Subject	Methods		Classes		Total
	During	After	During	After	
P01	92	2	77	77	248
P02	106	2	108	12	228
P03	118	71	40	83	312
P04	101	3	72	73	249
P05	137	29	107	15	288
P06	137	74	159	65	435
P07	33	78	34	39	184
P08	90	72	96	51	309
P09	28	69	49	79	225
P10	140	129	158	86	513
Total	982	529	900	580	2991

Table 3: Number of quality concerns found in code reviews.

Category	Method	Class	Total
Coding style violation	17	52	69
Bug	14	34	48
Missing test	6	11	17
Insuffic. exception handling	5	9	14
Inadequate comments	3	8	11
Other	1	2	3
Total	46	116	162

4.5 Data Mapping

Figure 3 illustrates some of the ratings and biometric data that we collected for participant P02 on class `ClassX.java` over the course of his/her work on a change task. During the depicted time period, P02 was interacting with `ClassX` seven times. At three points in time during the depicted period, the developer was prompted by our plugin to rate the perceived difficulty while working with this class. For the three ratings, the perceived difficulty changed from three to one to five. While the developer was working on this

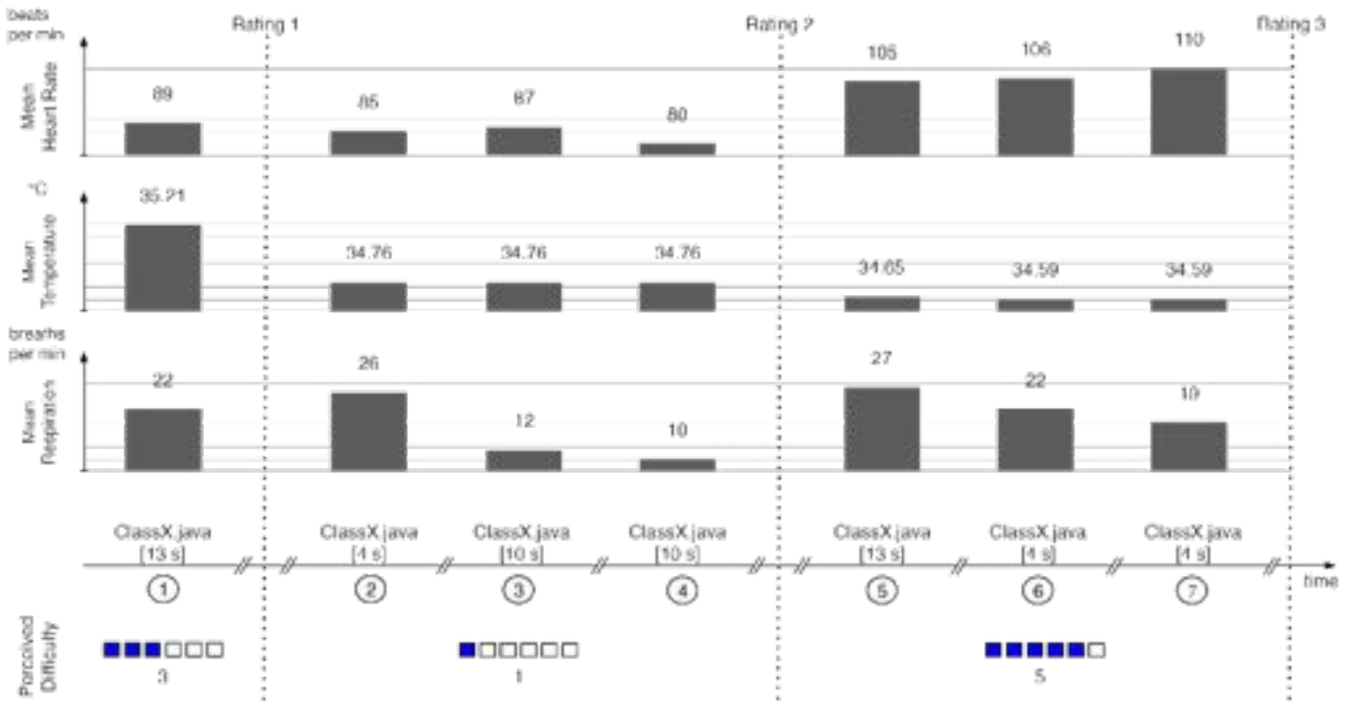


Figure 3: Exemplary perceived difficulty rating and biometric data (heart rate, skin temperature and respiration rate) over seven time periods during which participant P02 worked on the class `ClassX.java`.

class, we also captured the biometric measurements as described earlier on. A subset of these metrics is also depicted in Figure 3. For each rating by the developer on perceived difficulty during work, we associated the biometric measurements collected between the current and the previous rating. For the example shown in Figure 3, we only considered the biometrics captured in interval ① for 'Rating 1', the intervals ② - ④ for 'Rating 2' and intervals ⑤ - ⑦ for 'Rating 3'. In this example, there is a visible difference with the mean heart rate being rather low between 'Rating 1' and 'Rating 2' for which interval the class was perceived easy (1), and the mean heart rate between 'Rating 2' and 'Rating 3' when the class was perceived more difficult (5). For a developer's rating of perceived difficulty at the time of a commit, we associated all biometric measurements collected between the current and the previous commit.

For each code metric we captured for a code element and a given time frame—either between two ratings or between two commits—we calculated and collected the metric every time a developer interacted with the element and then calculated the mean over all interaction instances within the considered time frame.

5. ANALYSIS AND RESULTS

In the following, we address our research questions by presenting the analysis and results of the gathered data.

5.1 Perceived Difficulty and Quality of Code

Figure 4 depicts the distribution of collected difficulty ratings. Overall, only very few code elements (3.0%) that developers worked with were perceived as difficult or very difficult, while most (69.3%) were perceived as very easy or easy. To investigate whether and how the perceived difficulty of a code element changes over time, we analyzed a developer's

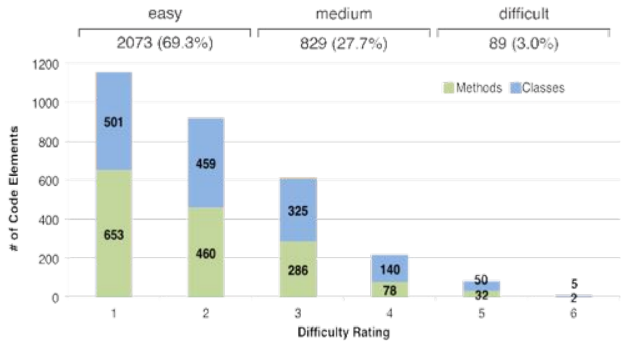


Figure 4: Distribution of developers' difficulty ratings of code elements.

difficulty ratings that we collected for the same code element during or at the end of a change task. While we did not collect multiple ratings for each code element during a change task due to the random selection process, we had 42 (±31.7) cases per developer in which we did. In 51.2% of these cases, the perceived difficulty changed between two consecutive ratings, with 43% of these cases in which the perceived difficulty increased. In most of these cases in which the perceived difficulty changed over the time a developer worked on a change task, code metrics did not change. For instance, the number of lines metric only changed in less than half, and McCabe's cyclomatic complexity only in less than a third of the cases. These results indicate that the perceived difficulty of a code element changes frequently over the course of a change task, and that these changes might often not be reflected in code metrics.

Table 4 provides an overview of the number of code elements that participants rated at commit time and the number of quality concerns that were found in these elements

Table 4: Quality concerns (QC) found in code elements during code reviews, grouped by perceived difficulty.

	Perceived difficulty					
	1	2	3	4	5	6
Methods						
# reviewed	245	176	90	15	3	0
# with QC	14	17	9	2	2	0
Classes						
# reviewed	174	189	145	47	22	3
# with QC	22	25	29	13	6	0

in the code reviews. As an example, from the 245 methods that were rated as being very easy, 14 (5.7%) were found to contain a quality concern during code review. The results show that **the more difficult a code element is perceived, the higher the likelihood of it containing a quality concern**, confirming our initial hypothesis. So while, for example, only 5.7% of the methods perceived as easy had quality concerns, 66.7% (2 out of 3) methods of the difficult elements that were rated as 5 had quality concerns.

5.2 Prediction of Code Difficulty and Quality

To answer our research questions, we performed a machine learning experiment. We chose machine learning, since it has been shown to be a good approach for finding links between low-level biometric data and high-level phenomena, such as perceived difficulty and quality concerns [8].

5.2.1 Machine Learning Approach

We conducted three kinds of predictions on two granularity levels—method and class level—each. In particular, we examined whether we can use machine learning to, first, predict a developer's perceived difficulty of a code element while working on a change task (RQ1), second, predict a developer's perceived difficulty of a code element after completing, *i.e.* committing, the work on a change task (RQ1), and finally, predict whether a code element contains a quality concern found in a code review (RQ2). Since we collected less than four data points for participant P01, P02 and P04 for perceived difficulties of methods after a commit and machine learners need a bigger sample size for reasonable results, we excluded these three participants from this specific prediction analysis.

For the machine learning classifications, we used Weka [33], a Java-based machine learning framework. For the classifier, we opted for a Random Forest learner [13] under the assumption that the non-parametric characteristics of decision trees [43] would fit our collected data, which often exhibited a non-parametric distribution, and because Random Forest learners can deal well with small sample sizes [58]. Studies have shown that for bug prediction based on code and change metrics, the learner should not have a big influence on the performance [30, 42].

We performed a leave-one-out evaluation for each participant separately. This means for each participant and prediction, we trained our classifiers in turn with all data points we captured, except one, and then used the remaining one as test set. We made sure that no identical code elements were in both, the training and the test set. For comparing biometrics with more traditional metrics, we performed each of the six (3 x 2) predictions for five different classifiers: a classifier based on biometric data, one based on code metrics, one on change metrics, one on interaction metrics, and one that combines all metrics.

5.2.2 Machine Learning Results

We split our results based on the outcome measure.

Perceived Difficulty (RQ1, RQ3, RQ4). Figure 5 summarizes the results of our machine learning experiments for predicting perceived difficulty. In particular, it presents Cohen's kappa [16] values for predicting a developer's perceived difficulty of code elements during (Figure 5a) and after finishing (Figure 5b) a change task. Cohen's kappa measures the agreement between the prediction and the ground truth, taking into account the agreements that might occur by chance. According to Landis and Koch [38], kappa values from 0 to 0.2 can be considered as slight, from 0.21 to 0.4 as fair, from 0.41 to 0.6 as moderate, from 0.61 to 0.8 as substantial, and from 0.81 to 1 as almost perfect agreement. For comparison reasons, we also added the value for a naive predictor that always predicts the most dominant class but never any other one. To be of practical value, our biometric classifier should be able to outperform this naive predictor.

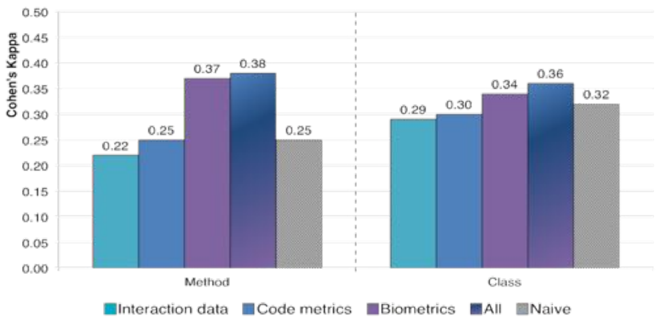
For three out of the four cases (see Figure 5) the biometric classifier outperforms the classifiers that are based on interaction metrics and on code metrics, as well as it outperforms the naive classifier on average by more than 26%. Only a classifier that combines all metrics, including biometrics, achieves better results in these cases, and only for the case of predicting the difficulty of classes after a change task is the biometric classifier worse than a naive one predicting only a dominant class. These results demonstrate the potential that biometrics have in particular for predicting the perceived difficulty of a code element online, while the developer is still working on the change task.

For a more detailed analysis of the results, we chose one case and depicted the confusion matrix for the perceived difficulty prediction on method level in Table 5. The matrix shows that in most cases, the predicted difficulty value (1 to 6) is only slightly off of the real value. Finally, Table 6 presents the percentage of correct predictions of the biometric classifier for each participant. These results illustrate the individual differences in the accuracy of predictions. For instance, some participants, such as P10, have a high accuracy for all predictions, while others, such as P07, have a high accuracy for some but not all predictions.

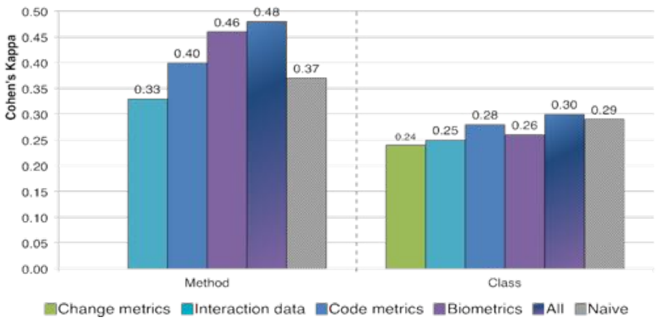
Quality Concerns (RQ2 & RQ3). When predicting whether a code element contains a quality concern (QC) or not (no QC), a biometric classifier performs best over all and even outperforms a classifier that combines all metrics. The top part of Table 7 presents the results in terms of precision and recall. We chose precision and recall instead of F-score to highlight the tradeoff between the two. The biometric classifier is able to correctly identify 17 out of 44 (38.6%) code elements with a quality concerns on method level and 38 out of 95 (40%) on class level. At the same time, the precision is not very high with 13.0%, respectively 22.0%, and further research is needed to examine this in more detail.

Table 8 provides more details on the percentage of correct classifications for each quality concern category. The data reveals that, with the exception of the "Other" category, the level of correctness is in a similar range for each category. Particularly interesting is the "Bug" category that shows that our biometric classifier is able to identify half of all bugs found in code reviews.

Within vs. Across Participant (RQ4). To investigate how sensitive the biometrics are to an individual, we per-



(a) During change task



(b) After change task (at commit)

Figure 5: Cohen's Kappa for predicting perceived difficulty for class and method level.

Table 5: Confusion matrix for perceived difficulty prediction on method level during the work on a change task. Each cell contains values from each predictor in the order of: all / biometrics / code metrics / interaction metrics / change metrics.

Actual	Prediction 1	2	3	4	5	6
1	328/310/284/267/-	59/63/88/87/-	15/27/23/44/-	5/8/9/7/-	0/0/4/3/-	1/0/0/0/-
2	91/83/95/92/-	149/158/127/113/-	43/36/52/64/-	1/7/7/10/-	0/0/3/5/-	0/0/0/0/-
3	49/54/55/56/-	67/58/64/67/-	67/70/57/67/-	8/10/14/6/-	5/4/6/0/-	0/0/0/0/-
4	15/13/21/11/-	13/16/17/19/-	19/19/16/20/-	13/13/7/12/-	3/2/2/1/-	0/0/0/0/-
5	7/4/13/13/-	3/5/3/6/-	7/4/9/6/-	1/2/3/3/-/	11/13/0/1/-	0/1/1/0/-
6	0/1/1/0/-	0/0/0/1/-	1/0/1/1/-	0/0/0/0/-	0/1/0/0/-	1/0/0/0/-

Table 6: Percentage of correct predictions per participant using biometrics.

Sub.	Difficulty (During)		Difficulty (After)		Quality Concerns	
	Method	Class	Method	Class	Method	Class
P01	64.1%	71.4%		64.9%		57.1%
P02	59.4%	53.7%		25.0%		83.3%
P03	55.1%	40.0%	53.5%	30.1%	97.2%	84.3%
P04	50.5%	52.8%		41.1%		43.8%
P05	43.1%	39.3%	37.9%	6.7%	96.6%	66.7%
P06	57.7%	49.1%	62.2%	30.8%	60.8%	58.5%
P07	33.3%	20.6%	61.5%	56.4%	66.7%	51.3%
P08	60.0%	37.5%	51.2%	39.2%	88.9%	70.6%
P09	53.6%	44.9%	68.1%	34.2%	53.6%	70.9%
P10	77.1%	81.0%	88.4%	86.1%	65.9%	77.9%
All	57.4%	53.3%	65.7%	46.9%	72.8%	66.0%

Table 7: Results for quality concern (QC) prediction within & across participants in % (P: precision, R: recall, ud.: undefined). Bold values accent the best result in each category.

Metric		Method				Class			
		QC		no QC		QC		no QC	
		P	**R**	**P**	**R**	**P**	**R**	**P**	**R**
Within	All	11.5	27.3	92.3	80.8	17.6	23.2	84.0	78.8
	Biometric	**13.0**	**38.6**	**93.1**	76.2	**22.0**	**40.0**	**86.0**	72.2
	Code	7.9	20.5	91.4	78.0	16.9	29.5	83.8	71.5
	Interaction	0.0	0.0	91.1	**94.6**	19.8	16.8	84.2	**86.6**
	Change					17.3	18.9	83.8	82.3
Across	All	**9.7**	**63.6**	**93.1**	45.2	17.5	**30.5**	84.1	71.8
	Biometric	8.3	56.8	91.3	41.8	15.4	22.1	83.3	76.3
	Code	8.1	50.0	91.2	47.9	**20.0**	20.0	**84.3**	84.3
	Interaction	0.0	0.0	91.4	**98.1**	16.7	3.2	83.6	96.9
	Change					ud.	0.0	83.6	**100.0**

Table 8: Percentage of correct quality concern predictions by category using a biometric classifier.

Quality Concern Category	**% Correct Method**	**Class**
Coding style	23.5	38.5
Bug	50.0	47.1
Missing tests	50.0	45.5
Insufficient exception handling	20.0	55.6
Inadequate comments	33.3	62.5
Other	100.0	0.0

formed a second machine learning experiment, in which we trained the classifiers not on each participant individually, but on data from all participants. We again used a leave-one-out approach to train the machine learning classifiers in turn for each participant, except one, and then used the remaining one as test set. We made sure that no code element was in both, the test and training set.

The results for predicting perceived difficulty either during or after a change task are very low. Cohen's kappa values were very close to or well below 0, showing that the predictive power of the classifiers is not any better than chance. For predicting quality concerns with a biometric classifier across participants, the results, however, are better and in some cases even outperform the prediction based on individual classifiers as presented in the bottom half of Table 7. The recall values on quality concern predictions on method level are significantly higher than the ones achieved by a within participant classification. However, this comes with a cost and the precision is generally lower and the recall for the code elements that do not contain a quality concern also decreased significantly. We hypothesize that the classifiers trained with data across individuals tend to predict more often that a code element contains a quality concern, since this case is represented in the training set more often, compared to the training set for each participant individually.

6. REPLICATION STUDY

Since there are many factors that might influence the study findings, such as the development process or the source code to name just a few, we performed a second smaller and shorter study. For this study, we collected similar but less data from five professional developers of a medium-sized software development company in Switzerland[4].

[4]For privacy reasons we are not able to disclose more specific information about the company.

Table 9: Number of collected data points per participant of the second study during and at the end of a change task.

Subject	**Methods**		**Classes**		**Total**
	during	**after**	**during**	**after**	
P11	132	-	165	101	398
P12	30	-	50	28	108
P13	23	-	60	60	143
P14	46	-	37	16	99
P15	7	-	16	9	32
Total	238	-	328	214	780

Study Method & Participants. For this second study, we were able to recruit five professional software developers, working at a medium-sized software development company in Switzerland. The five study participants worked in four different teams and each team worked on a different product. The study participants were all male, ranged in age from 25 to 30 years (∅=28.0, ±2.3) and had an average professional software development experience of 5.8 years (±2.5).

We followed the study method from our first study to collect metrics and outcome measures. All five participants used the SenseCore chest strap sensor for approximately one week. Over all participants, we collected data for a total of 25 work days (∅=5.0, ±1.6), including 2.8 GB of biometric data that consists of 11.6 million data points. Table 9 provides an overview of all data points collected during the second study. On average a study participant spend 3.4 minutes per method and 6.8 minutes per class between two consecutive difficulty ratings that occurred every 90 minutes.

Differences & Limitations. The teams in our main study and in the replication study followed a similar development process and developers worked on tasks with similar size. In contrast to the first study, the replication study only lasted one week due to time constraints of the participants. We also only had limited access to the code repositories and thus we were not able to collect perceived difficulty ratings on method level at the end of a change task and we were not able to collect any change metrics. Due to the lack of a code review process in the company, we were also not able to collect any data on quality concerns found in peer reviews. While these differences do not allow us to perform the same analysis, it still allows us to replicate some of the analysis in a vastly different setting. Especially in light of the effort and difficulty to find professional software developers that provide us access to their repositories and that are willing to wear biometric sensors for an extended period of time, we believe this is a reasonable step for an initial replication.

Machine Learning Predictions & Results. For the data collected during the second study, we extracted the same features from the data and performed the same leave-one-out within participant predictions as we did for the first study. Table 10 summaries the results of the predictions. In the second study, the biometric classifier outperforms classifiers based on interaction or code metrics in the 'after commit' case, *i.e.* after the code changes for a task were finished, with an improvement of more than 37%. In the other two cases for predicting difficulty during a change task, the classifier based on code metrics performs better, but the biometrics classifier is still significantly better than the naive classifier. Similar to the results of the first study, the classifier that incorporates all the different metrics is the best classifier.

Table 10: Cohen's kappa for perceived difficulty prediction for the second study.

Metric	**Method**	**Class**	
	(during)	**(during)**	**(after)**
Interaction	0.17	0.11	0.19
Code	0.25	**0.22**	0.27
Biometrics	0.20	0.16	0.37
All	**0.29**	**0.22**	**0.38**
Naive	0.06	0.14	0.07

In summary, the results of the second study provide initial evidence that we can replicate some of our findings from our main study, but not all. There are many potential reasons for the differences in findings, one of which is that we only collected about half the time of biometric data per code element and rating. Further studies are needed to investigate these aspects in more detail.

7. DISCUSSION

In this section we discuss the results of our study as well as their implications on practice and further research.

Predicting Code Quality Online. Our study is the first longer-term study in a real-world software development context with biometric sensors that provides evidence on the feasibility of using these sensors in the field. The results show that it is possible to predict quality concerns and perceived difficulty of code elements with higher accuracy than traditional metrics in most cases, even despite the noise in professional work environments. Biometrics, different to traditional metrics, allow for online—while the developer is still working on the code—measures, and thus, for example, to prevent bugs from ever being committed by focusing developers' attention on these parts, without requiring access to repositories. Biometrics also factor in developers' individual differences that are not captured by traditional metrics, and thus should provide more accurate results, in particular when they can be trained on each individual.

Our results also show that code elements that are perceived more difficult are also more likely to contain quality concerns. This adds to existing evidence that the difficulty a developer experiences when working on a code element can have a strong influence on the quality concerns the developer creates or adds when changing the code element for the task at hand. Consistently, our results suggest that it is possible to use biometrics not only for predicting perceived difficulty, but also quality concerns identified in peer code reviews.

While the precision for identifying quality concerns in our study could be higher, the fast technology advances leading to more accurate and less noise-sensitive sensors should soon lead to an increase in precision and the value of biometrics in this context. Also, we performed the data analysis for this study retrospectively, but the sensors we used already support real-time data transmission. Since the predictions only require short time windows of a few biometric features, almost instantaneous feedback should soon be possible.

While our smaller scale replication study provides evidence that some of our initial findings can be replicated in other settings, it did not confirm all of our findings. Even though, the biometric classifier still always outperformed the naive classifier, in two out of the three predictions, the biometric classifier was outperformed by a classifier based on code metrics. There are many potential reasons for this, *e.g.* the development phase, the source code structure, the developers' personalities, or even just the fact that the two studies were performed on different continents. Especially given the sensitivity of biometric sensors, further, longer term studies are planned to investigate these aspects in more detail.

Tool Support. Our results open up new opportunities for providing tool support. Since we are able to predict early on—while developers are still writing code—whether a code element contains a quality concern, we might be able to help developers and prevent them from ever submitting code with quality concerns to the repository. This could be done by highlighting the affected code element(s) to the developer before s/he commits them and suggesting to spend additional time reviewing. Similarly, one can use this information to suggest which parts of the code might benefit most from a peer code review and prioritize them. Biometric data could also be used to detect when a developer is experiencing difficulties within the code and to provide interactive and immediate feedback ranging from a recommendation to talk to a co-worker to taking a break and continuing later on.

While the study results show that it is possible to detect when a developer experiences difficulties and determine the corresponding code elements to be able to provide the discussed tool support, more research is needed to assess how to best present this information to developers, especially without creating frustration. Also, to provide this kind of tool support the biometric data needs to be collected continuously and transferred in real-time which poses challenges due to sensor invasiveness and more as discussed below.

Challenges. Biometric sensors that have the capability to collect the fine-grained data needed for the kind of study presented here are still under development and pose several challenges due to their usability, invasiveness and the data sensitivity. For our study, we always made sure to have one researcher on site to support participants and we chose sensors that could be worn for several weeks without being too invasive. With the recent advances in sensor technology, the physical invasiveness will decrease even further in the near future. At the same time, more privacy and ethical concerns have to be addressed and investigated, especially since these sensors can be used to collect huge amounts of very sensitive, health-related data. For all these reasons, recruiting study participants who are willing to wear such sensors for weeks and agree to collect a lot of personal data was also very tedious and time consuming, but in the near future, people might almost automatically collect similar data when wearing watches, such as the Apple Watch [4].

8. THREATS TO VALIDITY

There are several threats to the validity of our study.

External Validity. The generalizability of our findings is limited in many ways, such as the limited number of participants and companies in our study, the focus on Java code and the use of the Eclipse IDE, or the limited number of code elements developers work with and perceive as difficult or very difficult. We tried to mitigate this risk by replicating our initial study and also by collecting data from professional developers in the field, working in different teams and even companies, over a longer period of time and on industrial project code. However, due the broad spectrum of software development differing in aspects, such as the development process, the change task size, the programming languages, the team size, and the development phase the team is in to name just a few, further studies are needed to investigate their implications on the use of biometrics.

Internal Validity. In one part of this study, we used biometrics to predict the quality and difficulty of code elements. A threat to the study is that the data captured with biometric sensors might be affected by other aspects than the perceived difficulty, such as the study participants' personality traits or their general stress level. To mitigate this risk, we used the fish tank videos to capture a baseline each day and normalize the data with it.

Construct Validity. Using the interaction log as an approximation of the code elements a developer might currently be thinking about and using this to segment the biometric data also poses a threat to the validity of our study. However, given the current technologies, this was the best approximation that was also feasible. Eye-tracking devices might provide even more accurate and richer data on which code elements a developer is looking at and thinking about as another study has shown [35], but eye-tracking devices are currently too expensive or invasive to be used in a long-term field study of this size. Another threat to validity is the use of developers' self-reports, since they might not always accurately represent their experienced difficulty. Finally, our comparison to traditional metrics is limited due to the lack of access to the necessary data and repositories, and future studies are needed to also examine other metrics linked to code quality, such as code churn between multiple versions.

9. CONCLUSION

There is a broad range of tools and research that focuses on the identification of code quality concerns. Yet most of these approaches only allow for a post-hoc assessment and do not take individual differences of developers into account, such as different expertise or experience. In this paper, we presented a first two-week field study on the use of biometric sensors to identify code quality concerns while a developer is working on the code. The results of our study are promising, suggesting that developers' biometrics can indeed be used to determine the perceived difficulty of code elements and furthermore to identify places in the code that end up with code quality concerns, such as bugs. A second smaller replication study we conducted also confirmed some of our findings on the automatic determination of difficult parts in the code. These results open up new opportunities to support developers when they are experiencing difficulties in the code and to fix quality concerns as early as possible, even right when they are being created. With the recent advances in biometric sensing technologies, and their decrease in invasiveness, we might soon be able to collect biometric data on each developer just like we are now already able to collect interaction data. However, this also opens up a discussion on privacy concerns and more research is needed to investigate a feasible solution.

Acknowledgments

The authors would like to thank the study participants. This work was funded in part by ABB and the SNF project "Essentials (People-centric Essentials for Software Evolution)" (project no. 200020_153129).

10. REFERENCES

[1] A. F. Ackerman, P. J. Fowler, and R. G. Ebenau. Software inspections and the industrial production of software. In *Proc. of Symp. on Softw. Validation*, 1984.

[2] E. H. Alikacem and H. Sahraoui. Generic metric extraction framework. In *Proc. of IWSM/MetriKon*, 2006.

[3] L. Anthony, P. Carrington, P. Chu, C. Kidd, J. Lai, and A. Sears. Gesture dynamics: Features sensitive to task difficulty and correlated with physiological sensors. *Stress*, 1418(360), 2011.

[4] http://www.apple.com/watch/.

[5] P. Ayres. Systematic mathematical errors and cognitive load. In *Contemporary Educational Psychology*, 2001.

[6] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proc. of ICSE*, 2013.

[7] R. Bednarik and M. Tukiainen. An eye-tracking methodology for characterizing program comprehension processes. In *Proc. of ETRA*, 2006.

[8] R. Bednarik, H. Vrzakova, and M. Hradis. What do you want to do next: a novel approach for intent prediction in gaze-based interaction. In *Proc. of ETRA*, 2012.

[9] G. G. Berntson, J. T. J. Bigger, D. L. Eckberg, P. Grossman, P. G. Kaufmann, M. Malik, H. N. Nagaraja, S. W. Porges, J. P. Saul, P. H. Stone, and M. W. van der Molen. Heart rate variability: origins, methods, and interpretive caveats. *Psychophysiology*, 34(6):623–648, 1997.

[10] B. W. Boehm. *Software engineering economics.* Prentice-Hall, 1981.

[11] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proc. of ICSE*, 1976.

[12] A. Bosu, M. Greiler, and C. Bird. Characteristics of useful code reviews: An empirical study at microsoft. In *Proc. of MSR*, 2015.

[13] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[14] S. Butterworth. On the theory of filter amplifiers. *Wireless Engineer*, 7:536–541, 1930.

[15] J. Carter and P. Dewan. Are you having difficulty? In *Proc. of CSCW*, 2010.

[16] J. Cohen. A coefficient of agreement for nominal scales. *Education and Psychological Measurement*, 20:37–46, 1960.

[17] A. M. Connor. Mining software metrics for the jazz repository. *Journal of Systems and Software*, 1(5):194–204, 2011.

[18] D. J. Cornforth, A. Koenig, R. Riener, K. August, A. H. Khandoker, C. Karmakar, M. Palaniswami, and H. F. Jelinek. The role of serious games in robot exoskeleton-assisted rehabilitation of stroke patients. In *Serious Games Analytics: Methodologies for Performance Measurement, Assessment, and Improvement.* Springer International Publisher, 2015.

[19] M. Crosby and J. Stelovsky. How do we read algorithms? a case study. *Computer*, 23(1), 1990.

[20] W. Cunningham. The wycash portfolio management system. *OOPS Messenger*, 4(2):29–30, 1993.

[21] B. Curtis, S. Sheppard, P. Milliman, M. Borst, and T. Love. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *Trans. on Software Engineering*, SE-5(2):96–104, 1979.

[22] R. G. Ebenau and S. H. Strauss. *Software Inspection Process.* McGraw-Hill, Inc., 1994.

[23] K. O. Elish and M. O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.

[24] http://www.empatica.com.

[25] http://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath-penned-these-words/.

[26] S. H. Fairclough, L. Venables, and A. Tattersall. The influence of task demand and learning on the psychophysiological response. *International Journal of Psychophysiology*, 56, 2005.

[27] J. Feigenspan, S. Apel, J. Liebig, and C. Kastner. Exploring software measures to assess program comprehension. In *Proc. of ESEM*, 2011.

[28] http://findbugs.sourceforge.net/.

[29] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliot, and M. Züger. Using psycho-physiological measures to assess task difficulty in software development. In *Proc. of ICSE*, 2014.

[30] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall. Method-level bug prediction. In *Proc. of ESEM*, 2012.

[31] http://www.niallkennedy.com/blog/2006/11/google-mondrian.html.

[32] R. Grady and T. Slack. Key lessons in achieving widespread inspection use. *Software*, 11(4):46–57, 1994.

[33] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[34] Y. Ikutani and H. Uwano. Brain activity measurement during program comprehension with NIRS. In *Proc. of SNPD*, 2014.

[35] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz. Tracing software developers' eyes and interactions for change tasks. In *Proc. of ESEC/FSE*, 2015.

[36] A. J. Ko and B. A. Myers. A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages & Computing*, 16(1):41–84, 2005.

[37] N. A. Kuznetsov, K. D. Shockley, M. J. Richardson, and M. A. Riley. Effect of precision aiming on respiration and postural-respiratory synergy. *Neuroscience letters*, 502(1):13–17, 2011.

[38] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[39] M. Lanza and R. Marinescu. *Object-oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems.* Springer, 2006.

[40] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In. Micro interaction metrics for defect prediction. In *Proc. of ESEC/FSE*, 2011.

[41] M. M. Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221, 1980.

[42] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Trans. on Software Engineering*, 34(4):485–496, 2008.

[43] O. Maimon and L. Rokach, editors. *Data Mining and Knowledge Discovery Handbook*. Springer, 2006.

[44] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proc. of ICSM*, 2004.

[45] S. McConnell. *Code complete*. Pearson, 2004.

[46] N. Moha, Y. Guéhéneuc, L. Duchien, and A. Le Meur. Decor: A method for the specification and detection of code and design smells. *Trans. on Software Engineering*, 36(1), 2010.

[47] R. Moser, W. Pedrycz, and G. Succi. Analysis of the reliability of a subset of change metrics for defect prediction. In *Proc. of ESEM*, 2008.

[48] S. C. Müller and T. Fritz. Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. In *Proc. of ICSE*, 2015.

[49] M. Munro. Product metrics for automatic identification of "bad smell" design problems in java source-code. In *Proc. of METRICS*, 2005.

[50] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proc. of ICSE*, 2005.

[51] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. of ICSE*, 2006.

[52] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proc. of ICSE*, 2008.

[53] 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 Proc. of ICSE*, 2014.

[54] D. Novak, J. Ziherl, A. Olenšek, M. Milavec, J. Podobnik, M. Mihelj, and M. Munih. Psychophysiological response to robotic rehabilitation tasks in stroke. *Trans. on Neural Systems and Rehabilitation Engineering*, 18(4), 2010.

[55] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. Detecting bad smells in source code using change history information. In *Proc. of ASE*, 2013.

[56] C. Parnin. Subvocalization - toward hearing the inner thoughts of developers. In *Proc. of ICPC*, 2011.

[57] https://pmd.github.io/.

[58] Y. Qi. Random forest for bioinformatics. In *Ensemble Machine Learning*. Springer, 2012.

[59] S. Radevski, H. Hata, and K. Matsumoto. Real-time monitoring of neural state in assessing and improving software developers' productivity. *Proc. of CHASE*, 2015.

[60] http://www.ifi.uzh.ch/seal/people/mueller/PredictCodeQualityWithBiometrics.

[61] P. Richter, T. Wagner, R. Heger, and G. Weise. Psychophysiological analysis of mental load during driving on rural roads - a quasi-experimental field study. *Ergonomics*, 41(5), 1998.

[62] P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: A case study of the apache server. In *Proc. of ICSE*, 2008.

[63] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello. Improving automated source code summarization via an eye-tracking study of programmers. In *Proc. of ICSE*, 2014.

[64] S. Schmidth and H. Walach. Electrodermal activity (EDA) - state-of-the-art measurements and techniques for parapsychological purposes. *Journal of Parapsychology*, 64(2), 2000.

[65] C. Setz, B. Arnrich, J. Schumm, R. L. Marca, G. Tröster, and U. Ehlert. Discriminating stress from cognitive load using a wearable eda device. *Trans. on Information Technology in Biomedicine*, 14(2), 2010.

[66] 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. of ICSE*, 2014.

[67] L. A. Sroufe and E. Waters. Heart rate as a convergent measure in clinical and developmental research. *Merrill-Palmer Quarterly of Behavior and Development*, 23(1):3–27, 1977.

[68] J. Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285, 1988.

[69] J. Sweller, P. Ayres, and S. Kalyuga. *Cognitive Load Theory*. Springer, 2011.

[70] E. van Emden and L. Moonen. Java quality assurance by detecting code smells. In *Proc. of WCRE*, 2002.

[71] J. Veltman and A. W. Gaillard. Physiological workload reactions to increasing levels of task difficulty. *Ergonomics*, 41(5):656–669, 1998.

[72] G. F. Walter and S. W. Porges. Heart rate and respiratory responses as a function of task difficulty: The use of discriminant analysis in the selection of psychologically sensitive physiological responses. *Psychophysiology*, 13(6), 1976.

[73] R. A. Weast and N. G. Neiman. The effect of cognitive load and meaning on selective attention. In *Annual Meeting of the Cognitive Science Society*, 2010.

[74] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, 13(5):539–559, 2008.

[75] G. F. Wilson. An analysis of mental workload in pilots during flight using multiple psychphysiological measures. *International Journal of Aviation Psychology*, 12(1), 2002.

[76] H. Zhang, X. Zhang, and M. Gu. Predicting defective software components from code complexity measures. In *Proc. of PRDC*, 2007.

[77] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Proc. of PROMISE*, 2007.