Biofeedback augmented software engineering: accurate monitoring of programmers' mental effort

Ricardo Couceiro
CISUC, University of Coimbra
Coimbra, Portugal
rcouceir@dei.uc.pt

Gonçalo Duarte
CISUC, University of Coimbra
Coimbra, Portugal
duarte.1995@live.com.pt

João Durães
CISUC, Polytechnic Institute of Coimbra
Coimbra, Portugal
jduraes@isec.pt

João Castelhano
ICNAS, University of Coimbra
Coimbra, Portugal
joaocastelhano@uc.pt

Catarina Duarte
ICNAS, University of Coimbra
Coimbra, Portugal
catarinaduarte86@gmail.com

Cesar Teixeira
CISUC, University of Coimbra
Coimbra, Portugal
cteixei@dei.uc.pt

Miguel Castelo Branco
ICNAS/CIBIT, University of Coimbra
Coimbra, Portugal
mcbranco@fmed.uc.pt

Paulo de Carvalho
CISUC, University of Coimbra
Coimbra, Portugal
carvalho@dei.uc.pt

Henrique Madeira
CISUC, University of Coimbra
Coimbra, Portugal
henrique@dei.uc.pt

***Abstract*—This paper presents emergent experimental results showing that mental effort of programmers in code understanding tasks can be monitored through HRV (heart rate variability) using non-intrusive wearable devices. Results suggest that HRV is a good predictor for cognitive load when analyzing code and HRV results are consistent with the mental effort perceived by programmers using NASA-TLX. Furthermore, code complexity metrics do not correlate entirely with mental effort and do not seem a good indicator of the subjective perception of complexity felt by programmers. This first results are presented in the context of the project BASE-Biofeedback Augmented Software Engineering, which is briefly sketched, and proposes a radical neuroscience enabled approach to introduce biofeedback in software development.**

Keywords—software faults, HRV, mental effort, complexity

I. Introduction

Software faults (i.e., bugs) remain as one of the most persistent problems of software quality. After decades of intensive research on software engineering and on software reliability, the "industry average is about 15 to 50 errors per 1000 lines of delivered code" (KLoC) [1]. Even when software is developed using highly mature processes, the deployed code still has high density of residual bugs, from 2 to 5 bugs per KLoC [2, 3]. In recent years, this problem has been getting worse, not only due to the constant pressure to shrink time-to-market and cost of software, but also because the code size has increased dramatically. More lines of code means more bugs. Unfortunately, no one knows exactly where they are in the code, when they will reveal themselves, and, above all, the consequences of their activation. With the huge dependence of our society on software, bad code quality due to residual bugs represent one of the most enduring and difficult technical challenges.

Although field studies analyzing the nature of real bugs are not abundant, existing reports [4, 5] on bugs found in deployed software in open source projects [4] and in large IBM systems [5] show that in spite of radically different development methodologies and technical cultures, the types of software defects classified using the Orthogonal Defect Classification in these two studies are impressively similar [4]. The common element is obviously the human factor, suggesting that humans tend to err in similar ways and originate a limited set of software fault types. In fact, the field studies [4, 6] suggest a "normalization" of bug types due to the human element and identify a top-N of most common bug types, showing that only 18 bug types (i.e., code constructs types) are responsible for 67.6% of all bugs found in real products (note that "same bug type" does not mean "same bug"). This evidence that programmers tend to fail in similar ways, according to a small number of bug types, suggests the possibility of monitoring programmers to identify programmers' cognitive conditions that may precipitate coding bugs or bugs escaping human attention.

In this paper we show the first results of a large scale experiment using co-registration of several bio signals to study in detail programmers' mental effort monitoring in code understanding tasks, as part of a comprehensive interdisciplinary project named BASE [1] - Biofeedback Augmented Software Engineering (briefly described in section II). The key idea of BASE is to research software faults from neuroscience perspective to find the brain mechanisms involved in software error making/discovery (focusing on the Top-N of most common bug types) and the correlated psychophysiological manifestations that can be captured by wearable devices compatible with typical software development environments. The goal is to enhance software development paradigms through the introduction of a strong new element: the programmers' biofeedback.

The first results show that it is possible to measure programmers' mental effort with high confidence, while programmers read and understand different code units, through real-time monitoring of bio signals such as HRV (heart rate variability). The discussion also shows the relationship between the mental effort measured using HRV and both the software complexity metrics of the different code excerpts and the subjective mental effort perceived by the programmers using NASA-TLX (Task Load Index)[2].

The next section presents the background of our research and related work, Section III describes the elements of the experiment and protocol, Section IV presents the methodology used in the data analysis, Section V presents and discusses the results and Section VI concludes the paper.

II. Background and Related Work

Software development is essentially a human activity. In fact, in spite of some interesting examples of automatic generation of executable code from high-level specifications (but specifications can be faulty as well), most of the

1 BASE, POCI - 01-0145 - FEDER- 031581, July 2018, December 2021.
2 NASA-TLX site: https://humansystems.arc.nasa.gov/groups/TLX/

software produced today is still the result of an intensive human made process. And people may fail while doing complex and abstract tasks such as requirement elicitation, functional specification, software architecture design, code development, and testing. Considering specifically code related activities, researchers have recently started looking at the cognitive process underlying code comprehension using functional magnetic resonance imaging (fMRI), near field infrared spectroscopy (fNIRS), and electroencephalography (EEG). Siegmund and others have shown that programmers strongly recruit brain regions related to language and working memory to understand source code and identify syntax errors using fMRI [7, 8]. Floyd and others studied code and natural-language text comprehension also using fMRI [9] and Nakagawa and others studied the mental execution of source code by programmers using fNIRS.

Our recent cognitive studies, also using fMRI, focused on the moment when a programmer finds a bug in the code [11], and a novel causal connectivity pattern associated to the "eureka" moment of bug intuition (and posterior bug confirmation) was discovered [12]. Additionally, the distinct role for the insula in software bug monitoring and detection was identified [12]. Importantly, the insula activity levels were critically related to the quality of bug detection and it was shown that the activity in this salience network region evoked by bug suspicion was predictive of bug detection precision. This was the first time a brain signal could be related with software skills on bug detection [12].

It is a well-known fact that physiological responses driven by the autonomic nervous system (ANS) can be monitored by low intrusive sensors. In fact, there are many commercial wearable devices that can monitor ANS driven response such as heart rate variability (HRV), breathing rhythm and electrodermal activity (EDA), including sophisticated wearable devices equipped with photoplethysmography, eye tracking with pupillography, and even wearable versions of electrocardiography (ECG). A recent study has shown that HRV can be used to predict code quality and help identifying difficulties experienced by developers while dealing with a given code part [13]. However, physiological sensors are very sensitive to many other causes of physical body response, totally unrelated to the software development activities, and should be used with care as single source of programmers' biofeedback.

The BASE (Biofeedback Augmented Software Engineering) project, in whose context the present results have been produced, has two specific research motivations:

- Focus on bugs, to identify the brain networks that can be associated with the conditions that may precipitate programmers making bugs or bugs escaping human attention. This is especially motivated by the empirical evidences that most of the bugs fall into a small number of bug types [4, 5, 6], thus it makes sense to investigate those error prone program constructs from a neuroscience perspective.
- Focus on the connection between brain mechanisms involved in software error making/discovery and the correlated psychophysiological manifestations driven by the ANS. This is important to define and validate models and algorithms capable of filtering out the (noisy) data gathered by wearable physiological sensors, to improve the accuracy in the identification of the conditions (and the corresponding code locations in the software under development) that may precipitate programmers making bugs or bugs escaping human attention.

The BASE project vision is to provide neuroscience enabled programmers' biofeedback through the use of wearable sensors to monitor accurately programmers' physiologic reactions related to potential bug conditions, as well as programmers' cognitive stress, mental effort, concentration, or attention shifts states during software development. This meta-data about the programmers' cognitive state while dealing with a given code unit can be recorded and linked to the related program code lines, which will allow several new and visionary features such as:

- **Online advice to programmers and testers**, which can cover a range of possibilities, from simple warning of software code areas that may need a second look at (to remove possible bugs) to more sophisticated programmers' support scenarios that consider the biofeedback metadata in conjunction with the complexity of the code handled by the programmer and her/his history of previous bugs (we call this **alter-pair programming**, as an analogy to the agile pair programming approach, but without the need of the second programmer of the pair).
- **Biofeedback improved models of bug density estimation and software risk analysis** through the use of code complexity metrics enhanced with programmer's biofeedback metadata that shows how complexity is really perceived by each programmer.
- **New biofeedback driven testing approaches** using the meta information about the programmers' cognitive and emotional states while working on the different portions of the code, combined with traditional complexity metrics, to guide the testing effort to the code units representing higher bug risk.
- **Programmers' friendly integrated development environments** with automatic warning/enforcement of programmers' resting moments, when accumulated signs of fatigue and mental strain show that not only the code quality is doubtful but, above all, programmers' mental well-being must be protected.
- **Biofeedback optimized training needs** through the creation of individual programmers' profiles to help define training plans based on the biofeedback metadata associated to individual programmers.

III. Experiment design and protocol

The specific goal of the present experiments is to investigate how mental effort in reading and understanding programs of different complexity can be measured by a set of sensors placed in the programmers (subjects) that participated in the experiment. We used the experimental setup of one of the experiment campaigns of the BASE project, which covers a comprehensive set of sensors including EEG with a 64-channel cap, ECG, EDA, and eye tracking with pupillography. The signals from all these sources are synchronized in a common time base to allow consistent cross analysis. This setup allows several studies, currently under development (including correlations between the EEG brain activity and the other physiological signals), but the first results discussed in this paper only focus on HRV (heart rate variability), obtained from the ECG. In a real software development scenario, HRV can be monitored by a smart watch in a totally non-intrusive way.

The experiment included 26 participants and all the subjects had experience in the Java programming language, selected for this study. The screening process included an interview to assign a programming level skills in Java to each participant as Intermediate, Advanced or Expert.

The study was approved by the Ethics Committee of the Faculty of Medicine of the University of Coimbra, in accordance with the Declaration of Helsinki and all experiments were performed in accordance with relevant guidelines and regulations. Informed consent was signed by all participants. The anonymized data from this research are available upon request to the corresponding author.

We selected 3 small Java programs having different complexity. The programs were carefully designed to keep consistency in the programming style and to avoid that math or algorithm pose extra difficulties to the participants, not directly related to the code complexity. Program C1 counts the number of values existing in a given array that fall within a given interval using a straightforward loop. Program C2 multiplies two numbers using the basic algorithm where every digit from one number is multiplied by every digit from the other number, from right to left. The numbers are given as strings and the algorithm includes converting the strings to byte arrays. Program C3 seeks occurrences of an integer cubic array inside a larger cubic array, trying to find the larger occurrence of the smaller one inside the larger one. The algorithm for this program has a high cyclomatic complexity having many nested loops.

Table I presents a summary of the complexity metrics of the three programs. In C1 and C3 the algorithm is coded in one function, while in C2 the algorithm is spread across two functions, possibly making C2 easier to read than C3, in spite of both having a comparable number of lines of code.

TABLE I. PROGRAMS C1, C2 AND C3 USED IN THE EXPERIMENTS

Prog.	Lines of code	Nested Block Depth	No. params.	Cyclomatic complexity
C1	13	2	3	3
C2	42 (12+30)	3	3	4
C3	49	5	4	15

All the participants performed the experiments in the same room, without distractions, noise or presence of people unrelated to the experiments. The protocol includes the following steps, performed on the screen of a laptop:

1. Empty grey screen with a black cross in its center during 30 seconds. It serves as baseline phase in which the participant does not perform any activity.

2. Screen with a text in natural language to be read by the participant (60 seconds max.). This step serves as reference activity (different from code understanding) for the purpose of data analysis.

3. Empty grey screen with a black cross for 30 seconds.

4. Screen displays the code in Java of the program to be analyzed for code comprehension (C1, C2, C3). This step last 10 minutes maximum.

5. Empty grey screen with a black cross for 30 seconds.

This protocol is run 3 times for each participant, and in each run a different program (C1, C2, C3) is used in step 4. At the end of each run (i.e., after step 5) each participant fills in two brief surveys: one related to his/her understanding of the program C1, C2 or C3, and a second survey following NASA-TLX to assess the subjective mental effort perceived by each participant in the code comprehension.

IV. METHODS AND ANALYSIS PROCEDURE

Cognitive processes influence the Autonomic Nervous System (ANS), which is responsible for regulating cardiovascular system, among many other tasks. One of the most important non-invasive markers used to access the regulation mechanisms of the ANS over the cardiovascular system is the HRV, which is based on the evaluating of changes of time periods between consecutive cardiac cycles and is quantified from the analysis of the electrocardiogram.

A. Feature extraction and transformation

In this study, the sympathetic and parasympathetic activity of the ANS was assessed using both time and frequency domain HRV analysis [14] in a 60 sec sliding window, shifted by 5 sec increments. From this analysis, 6 features were extracted from the time domain and 17 features were extracted from the frequency domain, in a total of 23 features. Due to the inter-subject variations of these features, features were normalized in each trial according to the mean/median (defined by a one-sample Kolmogorov-Smirnov test) of the respective feature in the resting phase.

In order to increase the amount of information extracted from each phase (“resting” or “mental effort”) of the trial, the 23 extracted features were transformed using the following 5 operators: mean, median, standard deviation, maximum and minimum, resulting in 5 x 23 features transformations (115).

B. Feature space reduction and classification

To increase the interpretability of the classification model and to improve the efficiency of the training process and its generalization capability, the most important feature transformations were selected using the normalized mutual information feature selection (NMIFS) algorithm [15]. In this algorithm the objective is to create a subset containing the most relevant features for the problem and simultaneously the least redundant ones. The discrimination of the various predefined classes was performed using a Support Vector Machine (SVM) model. In this step, a C-SVC [16] algorithm with a radial basis function kernel was adopted.

The proposed methodology was validated using a 13-fold cross-validation scheme. Here, the global dataset was randomly partitioned in 13 equal size subsets and 12 subsets are used for training and the remaining subset is used for testing the classification model. This process is repeated 13 times corresponding to each of the 13 subsets.

V. RESULTS AND DISCUSSION

The signals from the 26 subjects were analysed using the aforementioned methodology and each segment of data was classified and compared to the previously defined classes: **1)** R - segments where the subjects were reading a text in natural language or looking at the calibration cross; **2)** C - segments where the subjects were analysing code, C1, C2, C3; **3)** C1-segments where the subjects were analysing code

C1; **4)** C2 - segments where the subjects were analysing code C2, and; **5)** C3 -segments where the subjects were analysing code C3. Based on the abovementioned classes, 7 case studies have been defined: **1)** R vs C; **2)** R vs C1; **3)** R vs C2; **4)** R vs C3; **5)** C1 vs C2; **6)** C1 vs C3; **7)** C2 vs C3.

For each of the 7 case studies, a classification model ($CM_i = \{CM_1,\ldots,CM_7\}$) was developed and validated using the before mentioned cross-validation scheme. The performance of the models was evaluated by the average and standard deviation (avg ± std) of the following metrics: sensitivity (**SE**) and specificity (**SP**). The sensitivity (or recall) is the ability of the classifier to detect the true positives (TP) and is defined by TP/(TP+FN), where FN are the false negatives. The specificity (or selectivity) is the ability of classifier to detect the true negatives (TN) and is defined by TN/(TN+FP), where FP are the false positives

Table II presents the results achieved by the developed classification models in each of the 7 case studies.

TABLE II. PERFORMANCE OF THE SVM CLASSIFIER (SE AND SP) FOR THE 7 CASE STUDIES USING 13-FOLD CROSS VALIDATION METHOD

	CM_1: R vs C		
SE	0.97±0.06		
SP	1±0		
	CM_2: R vs C1	**CM_3: R vs C2**	**CM_4: R vs C3**
SE	0.88±0.22	1±0	0.96±0.14
SP	0.92±0.28	0.96±0.14	1±0
	CM_5: C1 vs C2	**CM_6: C1 vs C3**	**CM_7: C2 vs C3**
SE	0.96±0.14	0.96±0.14	0.46±0.38
SP	0.81±0.25	0.85±0.24	0.46±0.38

We observe that the model CM1 was able to discriminate the resting phase from the code analysis phase (including all the 3 types of code) with a sensitivity of 97 ± 6% and a specificity of 100%, showing that the increase of cognitive effort during the code analysis tasks was clearly captured by the HRV derived features.

When inspecting the capability of the classification models to discriminate between the resting phase and each of the code analysis phases (C1, C2 and C3), it is possible to conclude that each of the three models (CM2, CM3 and CM4) performed very well, with specificity and sensitivity above 88%. Here, the worst result was achieved in case study 2, i.e. in the discrimination between the resting phase and the analysis of code C1. Due to the low complexity of C1, it was expected that the increase in the cognitive effort of the subject would be lower, and therefore more difficult to distinguish than the remaining and more complex codes.

The differentiation between the different types of code analyzed by the subjects was also evaluated in this study. Here, the classification models (CM5 and CM6) performed with sensitivities and specificities above 81%, showing that the models were able to distinguish the analysis of C1 from the analysis of C2 and the analysis of C1 from the analysis of C3. On the other hand, the capability of the model CM7 to distinguish the analysis of C2 from the C3, was 46 ± 38%, showing a much lower discrimination capability. From these results, it is possible to conclude that using the extracted HRV measurements it is possible to observe the influence caused by the cognitive effort in the ANS, which is then captured by the developed models. These changes in the ANS are more clear in C2 and C3 than in C1. Although the complexity of C3 is higher than C2 according to the complexity metrics, this was not consensual among the subjects that analyzed the code. This fact is in agreement with the low performance of the model CM7.

We used an adapted version of the NASA-TLX test where the categories related to physical effort were removed (not relevant to the task being performed). Participants reported in the mental effort category (in a scale from 1 to 6) resulting in an average of 1.8 (relatively easy) for C1, 4.7 (quite difficult) for C2 and 4.8 (quite difficult) for C3. The mental effort reported by the participants is consistent with the HRV results and deviates considerably from the cyclomatic complexity measured from each program. According to cyclomatic complexity metric, program C3 is much more complex that C2 (and C1). However, both HRV and mental effort consciously perceived by participants place programs C2 and C3 very close in terms of difficulty to understand. This seems to confirm that complexity metrics alone are not enough to guide testing effort and biofeedback is a very promising research avenue.

ACKNOWLEDGMENT

BASE project, POCI - 01-0145 - FEDER- 031581.

REFERENCES

[1] Steve McConnell, “Code Complete: A Practical Handbook of Software Construction”, Microsoft Press, 2004.

[2] S. Shah, M. Morisio, M. Torchiano “The Impact of Process Maturity on Defect Density”, ACM-IEEE International Symposium on on Empirical Software Engineering and Measurement, 2012.

[3] N. Honda, S. Yamada, “Empirical Analysis for High Quality SW Development”, Ameri. Jour. Op. Research, 2012.

[4] J. Durães and H. Madeira “Emulation of SW Faults: A Field Data Study and a Practical Approach”, IEEE Transactions on SW Engineering, vol. 32, no. 11, pp. 849-867, November 2006

[5] J. Christmansson and R. Chillarege, "Generation of an Error Set that Emulates SW Faults", Proc. of the 26th International Fault Tolerant Computing Symposium, FTCS-26, Sendai, Japan, 1996.

[6] R. Natella, D. Cotroneo, J. Duraes, H. Madeira, "On Fault Representativeness of SW Fault Injection", IEEE Transactions on SW Engineering, vol.39, no.1, pp. 80-96, January 2013.

[7] 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”, Proc. of the 36th International Conference on Software Engineering - ICSE 2014.

[8] N. Peitek, J. Siegmund, et al., “A Look into Programmers' Heads”, IEEE Transactions on Software Engineering, August, 2018.

[9] B. Floyd, T. Santander, and W. Weimer, “Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise”, ICSE 2017, pp 175–186, Piscataway, NJ, USA, 2017.

[10] T. Nakagawa, Y. Kamei, et al., “Quantifying Programmers’ Mental Workload During Program Comprehension Based on Cerebral Blood Flow Measurement: A Controlled Experiment”, Proc. of ICSE 2014.

[11] J. Duraes, H. Madeira, J. Castelhano, C. Duarte, and M. C. Branco, “WAP: Understanding the Brain at Software Debugging. In Proc. Int’ Symposium Software Reliability Engineering, pp 87–92, IEEE, 2016.

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

[13] S. C. Müller and T. Fritz, “Using (Bio)Metrics to Predict Code Quality Online”, Department of Informatics, University of Zurich, Switzerland, Proc. of 38th IEEE ICSE, 2016.

[14] F. Shaffer and J. P. Ginsberg, "An Overview of Heart Rate Variability Metrics and Norms," Front Public Health, vol. 5, 2017.

[15] P. A. Estevez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized Mutual Information Feature Selection," Neural Networks, IEEE Transactions on, vol. 20, pp. 189-201, 2009.

[16] C.-C. Chang and C.-J. Lin, "LIBSVM : a library for support vector machines”, ACM Transactions on Intelligent Systems and Technology," vol. 2, pp. 27:1--27:27, 2011.