Clean and Dirty Code Comprehension by Eye-tracking Based Evaluation using GP3 Eye Tracker

Jozsef Katona

University of Dunaújváros, CogInfoCom Based LearnAbility Research Team
Táncsics M. 1/A, 2400 Dunaújváros, Hungary
E-mail: katonaj@uniduna.hu

Abstract: During the observation, analysis, and examination of cognitive processes, human-computer interfaces are increasingly becoming widespread. Programming could also be seen as a complex cognitive process. This study aims to examine the efficiency of the clean code paradigm and compares it to the dirty code produced without the principles formulated in this technique. In addition to the traditional knowledge level test and subjective judgment, the readability and comprehensibility of the implemented code were determined by analysing the heatmap and gaze route besides measuring and evaluating eye movement parameters. Based on the statistical evaluation, it can be stated that there is a significant difference in the average number of fixations, the average fixation time, and the average length of routes between fixations measured by studying two differently written source codes. This means that in the case of the clean code, significantly less and shorter information recording and processing were necessary to understand the code.

Keywords: eye-tracking; programming; clean code paradigm; dirty code

1 Introduction

Since every segment of our life is now supported by software, the quality and usability of software products have become paramount. During the observation, analysis. and examination of cognitive processes, human-computer interfaces are increasingly becoming widespread. Programming could also be seen as a complex cognitive process. In the spirit of usability, the field of Human-Computer Interaction (HCI) has evolved since the 1970s, while various software life cycle models have emerged due to designability and traceability. Countless variants of these models have emerged over the years, but since 2001, agile development has dominated. Laying down agile guidelines has led to the development of numerous methodologies (e.g., Scrum, Kanban, XP) that provide guidance (like clean code) to industry actors to create quality software products.

1.1 Theoretical Overview

The aim of the clean code paradigm applied in the field of programming is to support the executing of a software code base during a software implementation that facilitates easy overview and understanding as well as contributes to effective testing and development. To write codes with this structure, programmers need to be familiar with concepts as Don't repeat yourself (DRY) applicable for methods or Single Responsibility Principle (SRP) concerning classes. [1] Today, developments in the research field of Human-Computer Interaction (HCI) allow the observation and examination of various cognitive processes. [2, 3] Thus, by now, the clean code paradigm briefly described above can be examined by applying various HCI-based procedures. One of such analysis possibilities is the examination of the parameters of eye movement, with which the characteristics that can also be related to programming can be observed, the recorded results can be evaluated, and finally, connections can be determined.

The study [4] focuses primarily on the mechanism of reading source code. The method of reading a traditional text-based document and a source code describing the operation of software while observing, recording, and evaluating eye movement parameters were compared with the contribution of test subjects. The results of the research show that reading a source code is much less linear than that of a traditional text document, and experienced programmers read a code base less linearly than beginner programmers. Further research [5-8] shows that novice programmers spend much more time reading comments than their advanced peers to understand the code. The results of these researches can be related to the principles of clean code, because when compiling these types of code bases, the source file must be strived to be as a newspaper article, that is it should contain high-level concepts and algorithms, while details should be emphasized going downwards and the lowest level functions should be placed at the end. Informative nomenclature should be applied to minimize comments. [1] In the research [9] on source code review, the visual attention of the test subjects was examined in an industrial environment, where the recorded results showed that the test subjects with programming skills had more efficient eye movement features; for example, better code coverage, attention span and error lines as well as comments. These features are particularly important in effective error detection, i. e. in code review activity. In [10], the possibility of more effective education in programming is analysed and, through empirical research, the eye movement parameters of expert programmers are examined during modelling and debugging tasks. Other studies [11-15] focused on the planning phase preceding the implementation phase of the software development life cycle, where the intelligibility, arrangement, planning and application of Unified Modeling Language (UML) diagrams with eye-movement tracking technology were examined. In this study the effectiveness of the clean code paradigm through gaze tracking is aimed to be examined with the involvement of test subjects. In addition to examining eye movement parameters,

the two techniques were also compared in the form of a traditional test, with which the objective and subjective responses and opinions of the test subjects were also evaluated.

2 Research Goal and Applied Methodology

During the examination, four smaller source codes were applied; two of them were made with the clean code paradigm, while the other two ignored the methodology recommendations. To observe and examine various eye movement parameters, a Gazepoint GP3 eye-tracker hardware unit, and to record the metrics the OGAMA software package were applied.

The examinations were carried out using the source codes detailed in Chapter 2, applying the Gazepoint GP3 eye-tracker hardware unit and the OGAMA software package as well as by completing tests. The test subject had to observe a source code compiled on the basis of clean code paradigm and a dirty code-based code base. The first source code to be studied was selected randomly from the four available implementations. The second code base to be observed varied depending on whether the test subject had analysed a clean code or a dirty code during the first test. If the first examined source code was based on clean code paradigm, only a dirty code could be analysed during the second round. As the aim of the study was to compare the two techniques, no changes were made to the source code elements apart from the methodological differences, so to avoid that the results obtained in the first study influence the results of the second examination, different abstraction was applied. For example, if someone started with a User modelling source code in the first round, they could only continue with the Employee modelling code in the second round and vice versa. Overall, in the case of each test subject, the two techniques were compared at a different level of abstraction, thus avoiding the use of knowledge from the first test in the second test round. The test subjects had 120 seconds to study each code base and then they had to answer questions concerning the code. For displaying the code lines, an LG 22M45 type with1920x1080 resolution and 22” diameter monitor was used.

2.1 The Test Subjects

A total of 23 university students between the ages of 19 and 22 (M=20.78, SD=1.28) who declared themselves to be healthy, including 11 females and 12 males, participated in the study. The test subjects volunteered for the study, which had two conditions. The first one was the successful completion of the courses’ Introduction to Programming and Programming 1, as these subjects describe and demonstrate the skills that are essential for completing the task. The second one was the lack of knowledge required to implement clean code.

2.2 The Applied Test Source Codes

During writing the source code, it was taken into account that only elements familiar to test subjects from their previous studies would be used. In Figure 1, the UML class diagram of the source of the Acta.Clean.User project can be seen. The source code was made in compliance with the clean code paradigm, which includes, among others, the proper naming of classes, properties, variables, methods, etc. (using correct parts of speech and talkative identifiers, avoiding noise words, etc.), readable implementation of functions (the principle of single liability, maximum didactic, no side-effects), the avoidance of applying the switch expression (using polymorphism with overridden method), omitting comments (informative, applying expressive use of names), correct code formatting (newspaper metaphor, indent level less than 3). In total, in the case of the Acta.Clean.User project, 5 classes, of which one is an abstract and one is an enumerator were implemented. The total length of the Acta.Clean.User source code is 55 rows. The source code itself can be found in Appendix 1.

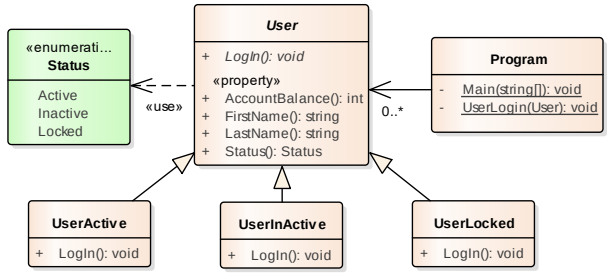


Fig. 1. The UML class diagram of the source code of the Acta.Clean.User project

Figure 2 shows the UML class diagram of the source code of the Acta.Dirty.User project. The source code ignores most of the recommendations listed in the clean code paradigm. The total length of the Acta.Dirty.User source code is 43 rows. The source code itself can be found in Appendix 2.

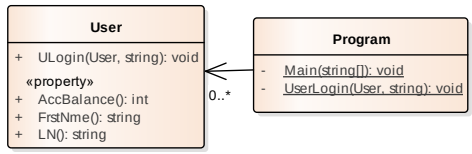


Fig. 2. The UML class diagram of the source code of the Acta.Dirty.User project

Figure 3 shows the UML class diagram of the source code of the Acta.Clean. Employee project. The source code was made following clean code paradigm, which includes those described in the Acta.Clean.User project. In this case, too, 5 classes, of which one abstract and one enumerator were implemented. The total length of the Acta.Clean.Employee source code is 57 rows. The source code itself can be found in Appendix 3.

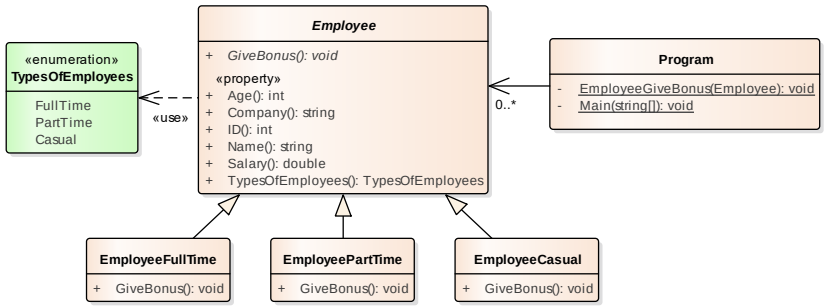


Fig. 3. The UML class diagram of the source code of Acta.Clean.Employee project

Figure 4 shows the UML class diagram of the source code of the Acta.Dirty.Employee project, which, like the Acta.Clean.User project ignores most of the recommendations listed in the clean code paradigm. The total length of the Acta.Dirty.Employee source code is 44 rows. The source code itself can be found in Appendix 4.

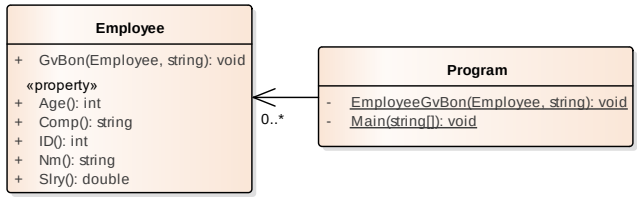


Fig. 4. The UML class diagram of the source code of Acta.Dirty.Employee project

2.3 The Gazepoint GP3 Eye-Tracker Hardware Unit

During the research, the Gazepoint GP3 research-grade eye tracker hardware unit (Fig. 5) was used to observe and examine eye movement parameters, which had been successfully applied in several previous research. [16-22] This unit can also be installed on the monitor and it detects and tracks gaze applying image processing by 60Hz sampling with infrared cameras.

Fig. 5. GP3 Eye Tracker

2.4 The OGAMA Software Package

During the examination, an open-source application, the OGAMA (OpenGazeAndMouseAnalyzer) was applied to record eye movement parameters observed and detected by the Gazepoint GP3 eye-tracker hardware unit, which supports other eye-tracking devices in addition to the hardware unit used in the research. The software package had been successfully used in several research applications. [22-25]

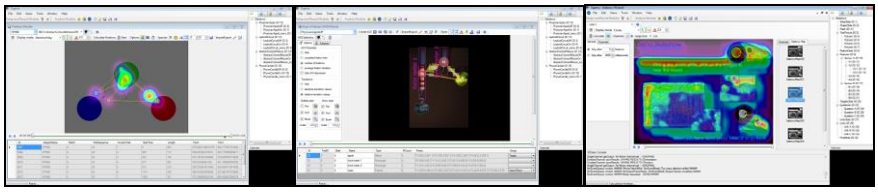


Fig. 6. Structure of OGAMA software

2.5 The Stages of the Examination

Before the research, the Gazepoint software package was downloaded and installed from the official manufacturer's site, and after successful installation, the GP3 eye-tracker hardware unit was connected via a USB port. After successful connection, the device was placed under the monitor about 65 cm from the eyes of the test subjects. When finding the right position, the gaze-date server was started and configured for real-time information retrieval and proper client server-based operation. After the server was functioning properly, the OGAMA software was launched and the Record Module was selected. After the successful launching and configuring of the OGAMA, the source code made with adequate technique and abstraction for the test case was opened in the Visual Studio Community development environment, where we tried to place it in a way that filled the screen as best as possible. Following proper preparation, the test subjects were provided all the necessary information related to the examination, and their important data, such as age or gender, were saved for further processing. After the successful data recording, in each test case, the calibration of the eye tracking device had to be done individually, during which the test subjects had to follow a circle moving their eyes from the left top of the monitor without moving their head. Calibration may have to be performed in the case of a test subject several times to achieve the best possible result. After successful and proper calibration, the test subject could begin to study the source codes. During the test, the eye movement parameters supported by the hardware unit and the OGAMA software were observed and recorded, and after the completion of the study, the data were saved in a database for further statistical evaluation. After successful data backup, the test subjects had to complete a test on the source code. With this method, we also tried to assess the

difference between the two techniques and to test the subjective opinions of the test subjects. A schematic diagram of the testing environment is shown in Fig. 7.

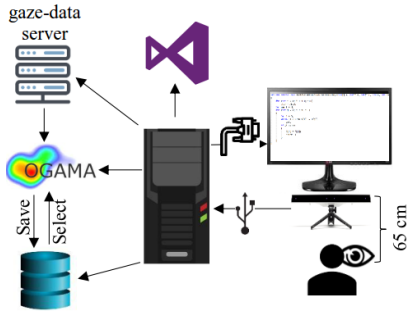


Fig. 7. A schematic diagram of equipment setup.

4 Results

The determination of the results started by evaluating the traditional tests. Each test consisted of 12 questions, identical on each test, of which 7 were used to check the readability and comprehension of the source code, and 5 questions were made to ask about the subjective opinion of the test subject.

4.1 The Evaluation of the Results of the Traditional Knowledge Level Tests

In the case of the source code based on the principles of clean code, all test subjects answered the questions on code readability and comprehensibility. The results show that most of them, five test subjects answered the fifth question, which asks how the whole code works, incorrectly, that is the 21.74% of the test subjects, while all of them could answer the fourth question, which asks some part of the code works, correctly. For the total sample, M=2.86, SD=1.77 incorrect answers were received, which corresponds to a total of M=12.42, SD=7.71 percent.

Even in the case of the dirty code test questions, all test subjects answered the questions on code readability and comprehensibility, were in the worst case 8, which is the 34.78% of the test subjects, answered to question 3 incorrectly, while one person gave an incorrect answer to question 2. Concerning the total sample, M=5.29, SD=2.29 incorrect answers were given, which corresponds to a total of M=22.98, SD=9.95 percent.

Table I. summary table shows the number and percentage of the incorrect answers in the knowledge level test regarding the clean code and the dirty code.

TABLE I.
A SUMMARY TABLE ON THE NUMBER AND PERCENTAGE ON THE INCORRECT ANSWERS/INCORRECTLY ANSWERING TEST SUBJECTS REGARDING THE CLEAN AND DIRTY CODE (*N*=23)

Clean Code				Dirty Code			
Min	Max	Mean	SD	Min	Max	Mean	SD
0	5	2.86 (12.42%)	1.77 (7.71%)	1	8	5.29 (22.98%)	2.29 (9.95%)

4.2 The Evaluation of Test Subjects' Subjective Opinion

A five-question survey was administrated to test subjects who participated in the examination to elicit their opinion on the clean and dirty codes, using a five-point Likert-type scale of "A": not at all; "B": slightly; "C": moderately; "D": pretty; "E": completely. Based on the above, five questions were formulated (Qs):

Q_1: How much did you feel reading an article while reading the code?

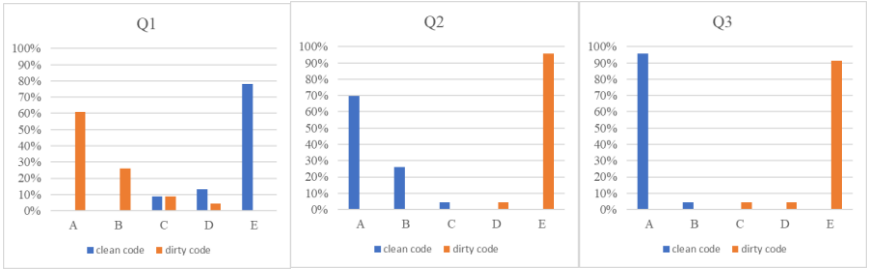
Q_2: How difficult was it to understand the source code?

Q_3: To what extent did you feel the need for comments?

Q_4: How uncertain was the code?

Q_5: How much did you feel the lack of precision?

According to the test subjects, the clean code is easier to read and better to understand without comments than the dirty code. Furthermore, the clean code was felt to be much more precise, causing much less uncertainty in their understanding. The evaluation of the results can be seen in Fig. 8.



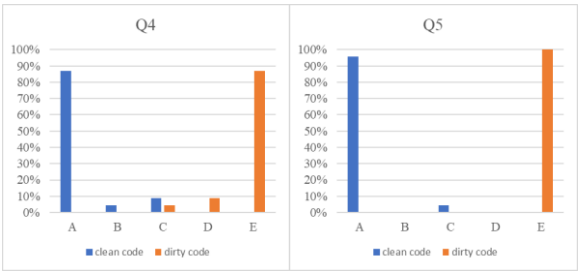


Fig. 8. Responses to questionnaire.

4.2 Evaluating the Parameters of Eye Movement

During the evaluation of the eye movement parameters, the heatmaps that the OGAMA software can generate in the Scanpath module were first examined along displaying the gaze route. After observing, examining, and evaluating the heatmaps, fixation numbers, average fixation durations, and Fixation Connections Lengths were also analysed and evaluated.

4.2.1 Evaluating the Recorded Heatmaps

Fig. 9. (1) and (2) show the heatmap and the route of gaze following the clean code paradigm, while Fig. 9. (3) and (4) shows a heatmap of the source codes and the route of gaze where these principles are disregarded, randomly selected but are generally characteristic of test subjects. The maps show that the code reading was non-linear and the variability, the return to previous code lines was much less observable at the clean code than at the dirty code, as according to the gaze routes, the test subjects returned to the beginning of the source file more times, presumably because of comprehension problems. All in all, the examination of the codes seems much more uncertain in the case of the dirty code and reflects that the test subject was unable to fully comprehend the text even after repeated reading.

The colours used on the heatmap have the following meaning:

- transparent area: observed and focused area not at all or only for a very short time.
- green: observed, focused area for a short time.
- yellow: observed, focused area for a medium-length time.
- red: observed, focused area for a longer period.
- yellow lines: the route of the gaze.

(1) (2) (3) (4)

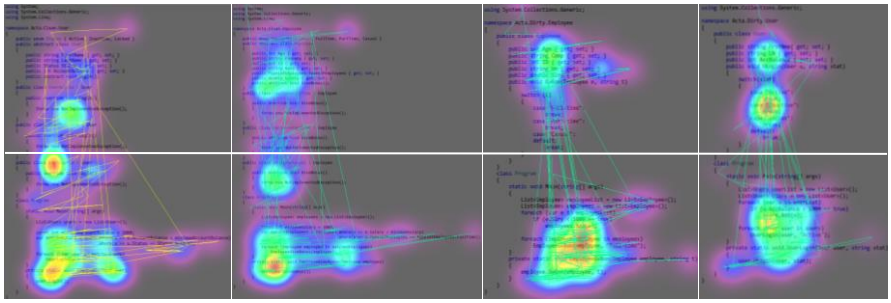


Fig. 9. The heatmap of a randomly selected test subject with gaze route following regarding Acta.Clean.User, Acta.Clean.Employee, Acta.Dirty.User and Acta.Dirty.Employee source codes.

Although the differences can be seen based on the heatmaps, they can only be quantified by using the Area Of Interest (AOI) tool of the software, which is used to link eye-movement measures to parts of the stimulus used. In this examination, the code was a single area.

4.2.1 The Evaluation of the Index Numbers of Eye Movement

The first evaluation was carried out on the interval scale with respect to the number of fixations. In the case of the source codes written in different ways, the Shapiro-Wilk test results applied during the examination of the normality of the data are not significant, in the case of the clean code it is ($D(23)=0.948$, $p=0.262$), in case of the dirty code it is ($D(23)=0.917$, $p=0.057$). As the data show normal distribution and we compared the fixation amount of the same test subjects in a group, the paired-samples t-test was applied to show that there is a significant difference in the mean number of fixations measured at the clean code and the dirty code ($t(22)=-3.528$, $p=0.002$ (2-tailed), $d=0.942$). The test subjects in case of the clean code showed on average fewer information captures ($M=382.29$, $SD=157.37$) than in the case of the dirty code ($M=520.61$, $SD=135.42$). The confidence interval for the mean of the fixations is shown in Fig. 10.

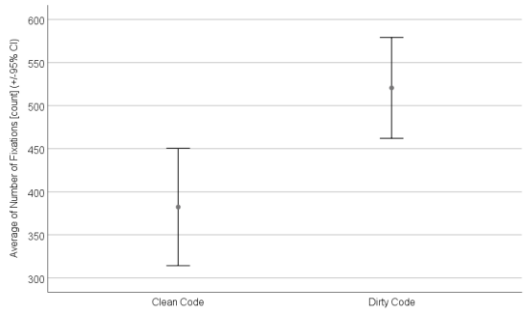


Fig. 10. The confidence interval for mean of the number of fixations.

The second evaluation was also performed on an interval scale with respect to the fixation time. In the case of differently written source codes, the Shapiro-Wilk test results used during the examination of the normality of data are not significant in case of the clean code, but in the case of the dirty code, they are significant. In the case of the clean code, it is ($D(23)=0.961$, $p=0.492$), while in case of the dirty code it is ($D(23)=0.777$, $p<0.001$). As the data do not follow a normal distribution in case of the dirty code and in one group the fixation time is compared at the same test subjects, the Wilcoxon-test was applied, according to which it can be determined that there is a significant difference between the average fixation time measured at the clean code and the dirty code ($T=37$, $Z=-3.072$, $p=0.002$ (2-tailed), $r=0.64$). The test subjects on average in case of the clean code spent less time studying the code ($Mdn=314.89$) milliseconds than in the case of the dirty code ($Mdn=335.04$) milliseconds. The distribution of the average fixation duration is shown in Fig 11.

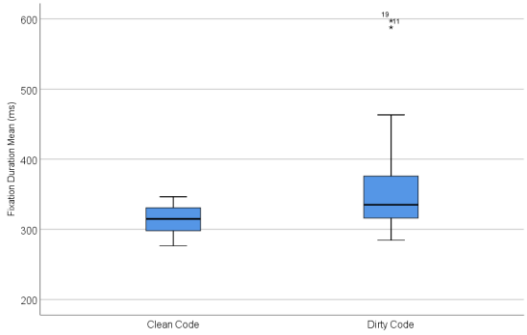


Fig. 11. The distribution of average fixation duration.

The third and final evaluation was carried out on an interval scale regarding Fixation Connections Length. In the case of the source code written in different ways, the Shapiro-Wilk test results used to test the normality of the data are not significant, in the case of the clean code it is ($D(23)=0.922$, $p=0.073$), while in the case of the dirty code it is ($D(23)=0.926$, $p=0.090$), as in the case of evaluating the number of fixations, a paired-samples t-test was applied, which shows that there is a significant difference in the average length of the routes measured between the fixations in the case of the clean code and the dirty code ($t(22)=-3.869$, $p<0.001$ (2-tailed), $d=0.995$). In the case of the clean code, the test subjects followed a shorter route between the fixations ($M=44779.354$, $SD=17876.352$) pixels than in case of the dirty code ($M=61377.328$, $SD=15371.635$) pixels. The confidence interval for mean lengths of fixation connections is shown in Fig. 12.

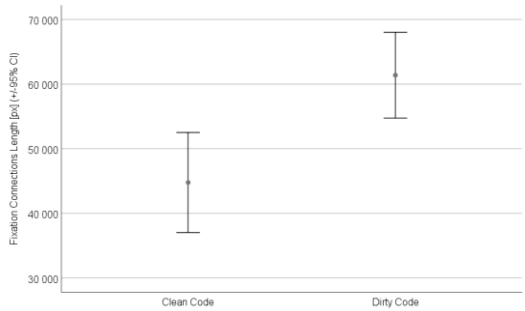


Fig. 12. The confidence interval for mean of the length of the fixation connections.

5 Discussion

After evaluating the results, it can be stated that the test subjects after studying the source files written using the clean code paradigm, achieved better results regarding the tests measuring the effectiveness of learning. As a consequence, it can be claimed that in the field of education, it is worthwhile to introduce and teach these types of codes from the beginning, in order to increase the efficiency of knowledge transfer and learning.

Based on the evaluation of subjective opinion questions, the test subjects felt that the top-down build of the source code following the principles of the clean code is well implemented and better suited to human information processing than the dirty code. In their opinion, the clean code besides easier readability is much easier to understand, and they did not feel the lack of comments, which cannot be stated in the case of the dirty code. In addition, they believe that uncertain codes often confusing programmers, in the case of the clean code were minimally or not at all present, which was attributed to accuracy.

Considering the results of the heatmaps observed by the OGAMA application, it can be concluded that in the case of the dirty code, the gaze is much more varied, and the test subjects focused more on parts of the source file that are less significant regarding code operation. Ultimately, this could also lead to the poorer results of the knowledge tests on understanding the dirty code, leaving them less time to study the more important code lines. However, with proper informative nomenclature, ignoring noise words, and applying top-down construction, that is, placing the more prominent parts at the end of the code, the source code reviewer can focus on the more important parts of the code line that affect the operation of the application. It is also confirmed by the results of the knowledge level measuring tests on the understandability of the clean code besides the heatmaps. The results examined in the research [4] show similarity with the results of the

current research since the reading of the codes was not linear, but the variability and the return to the previous code lines is much less observable in the case of clean code. This result may also mean that a novice programmer with more observable linear reading ability [4] can understand the clean code much more easily. This is also confirmed by the positive results of the clean code learning tests evaluated above. Overall, attention is less scattered with source code written using the clean code paradigm, and linearity is much more present in reading than in the case of the dirty code-based source files. These facts are confirmed by heatmaps and gaze routes generated from the recorded eye movement parameters.

After evaluating the eye movement parameters, it can be stated that there is a significant difference in the number of fixations, the duration of fixations, and the fixation connections length when comparing the two code types. This means that in the case of the clean code, significantly less and shorter information recording and processing were necessary to understand the code, and the distances between the fixation points were also significantly shorter. The importance of this is also related to the clarity of the code and its linear readability since the longer route means that test subjects had to return much more times to a more distant point of the code lines in order to understand its operation. Overall, the use of less informative names, the incorrect switch instruction, more frequent information recording, the longer information processing phase and the gaze route between longer fixations led to diverse attention and the more important parts from the aspect of the operation of the application were less focused. The effects of these can be seen in the results of the knowledge acquisition tests evaluated above.

Conclusions

Today, HCI-based technology is increasingly present in the analysis and examination of cognitive processes. In addition to general knowledge level questionnaires and subjective opinions, in this research, the efficiency of readability and understandability of the clean and dirty codes were analyzed with the eye movement parameters (fixation number, fixation duration, and fixation connections length) and the observation, examination, and evaluation of the generated heatmaps and the gaze route. The results show that source files written using the clean code paradigm are more readable and easier to understand than source codes ignoring this technique, which ultimately provides a more efficient testing opportunity and can greatly simplify application development and maintenance. As a result of the research, it can be stated that besides using subjective, traditional knowledge level tests, with the application of eye movement tracking systems, the understandability, readability and the quality of a source code can be examined and these can predict the difficulty of testing and further developing of the application. In education, pedagogical methodologies such as project-based education, in which students can learn in real-life contexts, [26] have been introduced due to the increasingly difficult and complex coding systems. Studies in [28, 29] primarily focus on developments that accompany the entire life cycle of a product. In addition to the modern pedagogical approaches

used today [27], HCI-based systems [30] could also function as a kind of learning [31] and education support system besides using modern learning environment [32-37] such as MaxWhere [38, 39]. In the future, in addition to the techniques and principles based on experience, eye movement parameters can serve as a support system for generating high-quality source code, which may become necessary to be applied in the field of industry besides education due to increasingly difficult and complex source files.

Acknowledgment

The project is sponsored by EFOP-3.6.1-16-2016-00003 founds, Consolidate long-term R and D and I processes at the University of Dunaujvaros. Also, thanks for the support of EFOP-3.6.2-16-2017-00018 „Produce together with the nature – agroforestry as a new outbreaking possibility" project.

References

[1] R. C. Martin: Clean code: a handbook of agile software craftsmanship, Pearson Education, 2009, p. 464

[2] P. Baranyi, A. Csapo, Gy. Sallai: Cognitive Infocommunications (CogInfoCom), Springer International Publishing Switzerland, 2015, p. 219

[3] P. Baranyi and A. Csapo: Definition and synergies of cognitive infocommunications, Acta Polytechnica Hungarica, vol. 9, 2012, pp. 67-83

[4] T Busjahn, et al.: Eye movements in code reading: Relaxing the linear order, In 2015 IEEE 23rd International Conference on Program Comprehension, 2015, pp. 255-265

[5] M. E. Crosby and J. Stelovsky: How do we read algorithms? a case study, Computer, vol. 23, no. 1, 1990, pp. 24-35

[6] M. E. Crosby, J. Scholtz and S. Wiedenbeck: The roles beacons play in comprehension for novice and expert programmers, Proceeding of Programmers 14th Workshop of the Psychology of Programming Interest Group, 2002, pp. 18-21

[7] T. Busjahn, C. Schulte and A. Busjahn: Analysis of code reading to gain more insight in program comprehension, Proceedings of the 11th Koli Calling International Conference on Computing Education Research ser. Koli Calling' 11, 2011, pp. 1-9

[8] T. Busjahn, R. Bednarik and C. Schulte: What influences dwell time during source code reading?: Analysis of element type and frequency as factors, Proceedings of the Symposium on Eye Tracking Research & Applications ser. ETRA '14. New York, 2014, pp. 335-338.

[9] R. K. Chandrika, J. Amudha, and D. S. Sudarsan: Recognizing eye tracking traits for source code review. In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2017, pp. 1-8

[10] S. Emhardt, C. Drumm, T. Van Gog, S. Brand-Gruwel and H. Jarodzka: Through the eyes of a programmer: a research project on how to foster programming education with eye-tracking technology, The 32nd Annual Conference of the Working Group Business Information Systems of the German-speaking Universities of Applied Sciences, 2019, pp. 42-49

[11] B. De Smet, L. Lempereur, Z. Sharafi, Y. Guéhéneuc, G. Antoniol and N. Habra: Taupe: Visualizing and analyzing eye-tracking data, Science of Computer Programming, vol. 79, 2014, pp. 260-278

[12] G. Cepeda Porras and Y. Guéhéneuc: An empirical study on the efficiency of different design pattern representations in UML class diagrams, Empirical Software Engineering, vol. 15, no. 5, 2010, pp. 493-522

[13] S. Jeanmart, Y.-G. Guéhéneue, H. A. Sahraoui, N. Habra: Impact of the visitor pattern on program comprehension and maintenance, Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 69-78

[14] S. Yusuf, H. H. Kagdi, J. I. Malefic: Assessing the comprehension of UML class diagrams via eye tracking, Proceeding of 15th IEEE International Conference on Program Comprehension ser. ICPC '07, 2007, pp. 113-122

[15] B. Sharif, J. I. Maletic: An eye tracking study on the effects of layout in understanding the role of design patterns, Proceedings of the 26th IEEE International Conference on Software Maintenance, 2010, pp. 1-10

[16] S. Seha, G. Papangelakis, D. Hatzinakos, A. S. Zandi and F. J. Comeau: Improving Eye Movement Biometrics Using Remote Registration of Eye Blinking Patterns, ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 2562-2566

[17] H. Zhu, S. Salcudean and R. Rohling: A novel gaze-supported multimodal human–computer interaction for ultrasound machines, International Journal of Computer Assisted Radiology and Surgery, 2019, pp. 1-9

[18] J. Meng, T. Streitz, N. Gulachek, D. Suma and B. He: Three-Dimensional Brain–Computer Interface Control Through Simultaneous Overt Spatial Attentional and Motor Imagery Tasks, in IEEE Transactions on Biomedical Engineering, vol. 65, no. 11, 2018, pp. 2417-2427

[19] J. Iskander, et al.: Age-Related Effects of Multi-screen Setup on Task Performance and Eye Movement Characteristics, 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018, pp. 3480-3485

[20] B. J. Edelman, J. Meng, N. Gulachek, C. C. Cline and B. He: Exploring Cognitive Flexibility With a Noninvasive BCI Using Simultaneous Steady-State Visual Evoked Potentials and Sensorimotor Rhythms, in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 26, no. 5, 2018, pp. 936-947

[21] S. Li and X. Zhang: Implicit Intention Communication in Human–Robot Interaction Through Visual Behavior Studies, in IEEE Transactions on Human-Machine Systems, vol. 47, no. 4, 2017, pp. 437-448

[22] A. Kovari, J. Katona and C. Costescu: Evaluation of Eye-Movement Metrics in a Software Debugging Task using GP3 Eye Tracker. Acta Polytechnica Hungarica, vol. 17, no. 2, 2020, pp. 57-76

[23] A. Voßkühler, V. Nordmeier, L. Kuchinke and A. Jacobs: OGAMA (Open Gaze and Mouse Analyzer): Open-source software designed to analyze eye and mouse movements in slideshow study designs, Behavior Research Methods, vol. 40, no. 4, 2008, pp. 1150-1162

[24] T. Ujbányi: Examination of eye-hand coordination using computer mouse and hand tracking cursor control, 2018 9th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), 2018, pp. 353-354

[25] T. Ujbanyi, J. Katona, G. Sziladi and A. Kovari: Eye-tracking analysis of computer networks exam question besides different skilled groups, 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), 2016, pp. 277-282

[26] G. Schrauf: Importance of project-based learning in software development, Computers & Learning, vol. 2, no. 1 2019, pp. 27-39

[27] S. Jambor: Educational methods based on student activity in vocational education, Transactions on IT and Engineering Education, vol. 2, no. 1, 2019, pp. 16-41

[28] R. Toth and R. Auer: Implementing and testing "Aubot" robot using self-study and collaborative learning strategies, Transactions on IT and Engineering Education, vol. 1, no. 1, 2018, pp. 25-41

[29] C. Csizmadia: Implementing a Java EE based information system as a student self-study task, Computers & Learning, vol. 2, no. 1, 2019, pp. 40-54.

[30] A. Skobrak: Direct hand-movement control in virtual space: a potential interface for virtual lab purposes, Transactions on IT and Engineering Education, vol. 2, no. 1, 2019, pp. 30-45

[31] Gogh E., Kovari, A.: Experiences of Self-regulated Learning in a Vocational Secondary School. Journal of Applied Technical and Educational Sciences, 9(2), 2019, 72-86.

[32] Gilányi A., Chmielewska K.: Educational Context of Mathability. Acta Polytechnica Hungarica, vol 15, no 5, 2018, pp. 223–237.

[33] Namestovski Z, et al. External Motivation, the Key to Success in the MOOCs Framework. Acta Polytechnica Hungarica, 2018, vol 15, no 6, pp. 125-142.

[34] Molnár G., Szűts, Z., Biró K.: Use of augmented reality in learning. Acta Polytechnica Hungarica, vol 15, no 5, 2018, 209-222.

[35] Orosz B. et al.: Visual, own device and experience-based educational methods and possibilities in VET, Proceedings of the 10th IEEE International Conference on Cognitive Infocommunications, 2019, pp. 517-520.

[36] Sik D. et.al.: Smart devices, smart environments, smart students – A review on educational opportunities in virtual and augmented reality learning environments, Proceedings of the 10th IEEE International Conference on Cognitive Infocommunications, 2019, pp. 495-498.

[37] Sik, D. et al.: Supporting Learning Process Effectiveness with Online Web 2.0 Systems on the basis of BME Teacher Training, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 337-340.

[38] A Horvath I., Sudar A.: Factors contributing to the enhanced performance of the Maxwhere 3d VR platform in the distribution of digital information. Acta Polytechnica Hungarica, vol 15, no 3, 2018, pp. 149-173

[39] Horvath I.: MaxWhere 3D Capabilities Contributing to the Enhanced Efficiency of the Trello 2D Management Software. Acta Polytechnica Hungarica, vol 16, no 6, 2019, pp. 55–71.

Appendix

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Acta.Clean.User
{
    public enum Status { Active, Inactive, Locked }
    public abstract class User
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Status Status { get; set; }
        public int AccountBalance { get; set; }
        public abstract void LogIn();
    }
    public class UserActive : User
    {
        public override void LogIn()
        {
            throw new NotImplementedException();
        }
    }
    public class UserInActive : User
    {
        public override void LogIn()
        {
            throw new NotImplementedException();
        }
    }
    public class UserLocked : User
    {
        public override void LogIn()
        {
            throw new NotImplementedException();
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<User> users = new List<User>();

            const int minimumAccountBalance = 1000;
            var matchedUsers = users.Where(u => u.AccountBalance < minimumAccountBalance)
                                    .Where(u => u.Status == Status.Active);

            foreach (User user in matchedUsers)
                UserLogin(user);
        }
        private static void UserLogin(User user)
        {
            user.LogIn();
        }
    }
}
```

Appendix 1. The source code of the Acta.Clean.User project

```
using System.Collections.Generic;

namespace Acta.Dirty.User
{
    public class User
    {
        public string FrstNme{ get; set; }
        public string LN { get; set; }
        public int AccBalance { get; set; }
        public void ULogin(User u, string stat)
        {
            switch(stat)
            {
                case "Active":
                    break;
                case "Inactive":
                    break;
                case "Locked":
                    break;
                default:
                    break;
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<User> userList = new List<User>();
            List<User> users = new List<User>();
            foreach (var u in userList)
                if (u.AccBalance < 1000 == true)
                    users.Add(u);

            foreach (User user in users)
                UserLogin(user, "Active");
        }
        private static void UserLogin(User user, string stat)
        {
            user.ULogin(user, stat);
        }
    }
}
```

Appendix 2. The source code of the Acta.Dirty.User project

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Acta.Clean.Employee
{
    public enum TypesOfEmployees { FullTime, PartTime, Casual }
    public abstract class Employee
    {
        public int Age { get; set; }
        public string Company { get; set; }
        public int ID { get; set; }
        public string Name { get; set; }
        public TypesOfEmployees TypesOfEmployees { get; set; }
        public double Salary { get; set; }
        public abstract void GiveBonus();
    }
    public class EmployeeFullTime : Employee
    {
        public override void GiveBonus()
        {
            throw new NotImplementedException();
        }
    }
    public class EmployeePartTime : Employee
    {
        public override void GiveBonus()
        {
            throw new NotImplementedException();
        }
    }
    public class EmployeeCasual : Employee
    {
        public override void GiveBonus()
        {
            throw new NotImplementedException();
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<Employee> employees = new List<Employee>();

            const int minimumSalary = 1000;
            var matchedEmployees = employees.Where(e => e.Salary < minimumSalary)
                                    .Where(e => e.TypesOfEmployees == TypesOfEmployees.FullTime);

            foreach (Employee employee in matchedEmployees)
                EmployeeGiveBonus(employee);
        }
        private static void EmployeeGiveBonus(Employee employee)
        {
            employee.GiveBonus();
        }
    }
}
```

Appendix 3. The source code of Acta.Clean.Employee project

```
using System.Collections.Generic;

namespace Acta.Dirty.Employee
{
    public class Employee
    {
        public int Age { get; set; }
        public string Comp { get; set; }
        public int ID { get; set; }
        public string Nm { get; set; }
        public double Slry { get; set; }
        public void GvBon(Employee e, string t)
        {
            switch (t)
            {
                case "Full-time":
                    break;
                case "Part-time":
                    break;
                case "Casual":
                default:
                    break;
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<Employee> employeeList = new List<Employee>();
            List<Employee> employees = new List<Employee>();
            foreach (var e in employeeList)
                if (e.Slry < 1000 == true)
                    employees.Add(e);

            foreach (Employee employee in employees)
                EmployeeGvBon(employee, "Full-time");
        }
        private static void EmployeeGvBon(Employee employee, string t)
        {
            employee.GvBon(employee, t);
        }
    }
}
```

Appendix 4. The source code of Acta.Dirty.Employee project