Code Beauty is in the Eye of the Beholder: Exploring the Relation between Code Beauty and Quality

Theodore Maikantis[1], Ilianna Natsiou[1], Christina Volioti[1], Elvira-Maria Arvanitou[1], Apostolos Ampatzoglou[1], Nikolaos Mittas[2], Alexander Chatzigeorgiou[1], Stelios Xinogalos[1]

[1] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

[2] Hephaestus Laboratory, School of Chemistry, Faculty of Sciences, Democritus University of Thrace, Kavala, Greece

Contact: a.ampatzoglou@uom.edu.gr

Software artifacts and source code are often viewed as pure technical constructs aiming primarily at delivering specific functionality to the end users. However, almost each line of a computer program is the result of software engineer's craftsmanship and thus reflects their skills and capabilities, but also their aesthetic view of how code should be written. Additionally, by nature, the code is not an artifact that is managed by a single person: the code is peer-reviewed, in some cases programmed in pairs, or maintained by different people. In this respect, the first impression for the quality of a code is usually a matter of "*reading*" the "*beauty*" of the code and then diving into the details of the actual implementation. This "*first-look*" impression can psychologically bias the software engineers, either positively or negatively and affect their evaluation. In this article we propose a novel code beauty model (accompanied with metrics) and empirically explore: (a) if different software engineers perceive code beauty in the same way; (b) if the proposed code beauty metrics are correlated to the perceived code beauty by individual software engineers; and (c) if code beauty metrics are correlated to software maintainability. The results of the study suggest: (a) that code beauty is highly subjective and different software engineers perceive a code chunk as beautiful or not in an inconsistent way; (b) that some code beauty metrics can be considered as correlated to maintainability; and therefore, the "*first-look*" impression might to some extent be representative of the quality of the reviewed code chunk.

***Keywords*:** Code Beauty, Code Quality, Technical Debt, eXplainable Artificial Intelligence, SHAP

1. Introduction

Modern software development methodologies rely heavily on the human-aspects of software engineering, dictating the use of practices (such as pair-programming (Begel and Nagappan, 2008) and code reviewing (Davila and Nunes, 2021)) that require the cross-checking of code by software engineers different than those that have originally written the code. On top of these inter-developer human-code interactions, maintenance tasks are in many cases assigned to software engineers, agnostically to who is the original contributor of the code. Reading, understanding, and changing the code that you have not authored is a task that is far more challenging than changing your own code (Davila and Nunes, 2021), promoting the understandability of code as an important factor for keeping maintenance cost (Chen et al., 2017; Al-Saiyd, 2017) at an affordable level. In the literature the "*wasting*" of maintenance effort, due to internal poor quality (such as readability and understandability) is communicated as Technical Debt (TD).

In the first minutes of code inspection, review, or maintenance, the software engineers' assessment on the anticipated effort for the task (and therefore the eagerness of the software engineer to start the task) can be biased by the "*first-look*" of the code (Chen et al., 2017; Al-Saiyd, 2017). This "*first-look*" is not related to the content or the quality of the code but is mostly biased by treating (looking at) the code in its entirety as an image or as a shape. Based on this assumption, in this paper, we aim at exploring if this inherent psychological bias, that can be aroused by the "*first-look*" at the code, can indeed mislead the software engineer, or if this first impression is

in the correct direction. Inspired by the study of aesthetics in other scientific disciplines (e.g., mathematics), we are exploring if "*code beauty*" can be correlated to the quality of the code, and more specifically to maintainability, which according to a recent literature review on TD interest is the most relevant factor for future TD payments (Arvanitou et al., 2020) (***High-Level Goal***).

The definition of "beauty" has been a matter of debate for many years and has been a distinct branch of philosophy dealing with the nature of art and beauty. We usually perceive something as beautiful when it is pleasing to the senses, especially eyesight. Many philosophers, psychologists and other scientists discuss if beauty can be objective, or if it is always subjective: "*Beauty is in the eye of the beholder*" (Duchess, 2008). Between these views, a common ground was found supporting that the aesthetic evaluation of an object is related to the observer's memories and feelings. Beauty is not only limited to objects or entities; it can also be observed in text and mathematical equations. There have been brain scans implying that seeing mathematical equations can sometimes evoke the same sense of beauty as masterpieces of painting and music suggesting that there is a neurobiological basis to beauty (Zeki et al., 2014). For example, Euler's identity is often cited as an example of deep mathematical beauty, due to its simplicity, involvement of only three arithmetic operations and five fundamental constants. Although constants such as e, π and i are complex concepts, they are beautifully linked by a simple and concise formula. Mathematicians usually describe a pleasing proof or technique as elegant, especially when it is concise and relies on a minimum number of previous assumptions and when it can be generalized to solve a variety of problems. As a result, we can assume that a similar case exists for source code (Oram and Wilson, 2007).

To this end, and to serve our high-level goal, in this study we propose a hierarchical "*Code Beauty Model* (CBM)*",* which first decomposes code beauty (1st level entity) into code beauty characteristics (2nd level entities), and subsequently links these characteristics to metrics (3rd level entities). Next, through empirical analysis, we explore if an assessment based on the CBM can be related to code quality, and more specifically maintainability. In this work, we investigate: (a) if software engineers have a common perception of code beauty (***subgoal-1***); (b) if CBM metrics can accurately capture the perceived code beauty (***subgoal-2***); and (c) if CBM metrics are correlated with maintainability (***subgoal-3***).

The rest of the paper is organized as follows: Section 2 presents related work: i.e., studies related to code beauty as well as studies that are related to the capture of aesthetics on Objects, Mathematics, and Text. In Section 3, we present in detail the Code Beauty Model, and in Section 4 the setup of our validation study. In Section 5 we report the results, which we discuss in Section 6. Finally, Section 6 concludes the paper.

2. Related Work

In this section we present related work. Initially, since there are no established code aesthetic metrics in the literature, first we discuss ***studies that are related to the capture of aesthetics on Objects, Mathematics, and Text***. Since code beauty and functionality are independent, we reuse measures for object, mathematical and text aesthetics. During the last centuries, several measures that define the beauty of objects have been proposed by scientists and artists. For example, balance was for the first time referred to as a beauty trait back in the 5th century when the famous Greek sculptor Polykleitos made his statue Doryphoros. A relation of beauty and harmony to symmetry was found during the Renaissance after the creation of the Vitruvian Man, a painting by the Italian artist Leonardo Da Vinci. Other traits found in the 19th century include quality and the form of the whole object which is affected by its color, proportion, and size (Santayana, 1955). The elements of regularity, mathematical harmony, order, and shortness along with symmetry, size and quality were also mentioned by artists (Loukaki, 2008). Another sector of beauty, i.e., mathematical beauty, introduced the elements of understandability and simplicity which turned out to be crucial for the beauty evaluation of mathematical formulae (Zeki et al., 2014).

These two factors were also mentioned by Wertheimer, i.e., the founder of the Gestalt theory (Wertheimer and Riezler, 1994), a theory which has been further extended by many other researchers.

Next, we focus on ***studies that are related to code beauty***. Oram and Wilson (2007) wrote a book for the importance of beauty in coding and the effect of beauty on its performance. In particular, the authors present a collection of papers from some well-known software engineers that reveal what they considered to be beautiful code in different programming languages in most cases. In this book, the beauty of the source code is related to performance, elegance, simplicity, and understandability of the final software product.

Coleman (2018) explored the correlation of code beauty with three maintainability indexes. To identify the relation between code beauty and maintainability, the author performs two testable hypotheses. He tests hypotheses on a corpus of 53,000 lines of code written by software engineers. Then, the author conducts a statistical correlation analysis with 33 experiments in total. Based on their findings, code beauty and maintainability seem to be two intricately connected aspects of software development. Beautiful code, characterized by clarity, simplicity, and adherence to best practices, inherently contributes to improved maintainability. When code is aesthetically pleasing, it becomes more readable and understandable for software engineers (Coleman, 2018). The relationship between code beauty and maintainability underscores the idea that writing elegant, clear, and well-structured code not only enhances the development process but also ensures that software remains adaptable and sustainable over time.

Eichelberger (2003) investigated the relation of beauty with the quality of UML diagrams. Specifically, he discussed various design criteria for UML class diagrams and emphasized the relation between the aesthetic quality of a diagram and the quality of the object-oriented design it represents. As a first step, the author described critics from the viewpoint of Human Computer Interaction to the UML notation and concluded that he does not require non-standard modifications to the UML notation guide. Then, the author listed relations between design and layout quality. The author analyzed object-oriented design metrics (e.g., number of children, depth of inheritance tree, size, etc.) leading to a formalized judgement of UML class diagrams with respect to object-oriented aspects. Then, Eichelberger (2003) presented a set of aesthetic criteria with respect to these results and discussed the realization of his proposal.

Finally, Aldenhoven and Engelschall (2023) highlight the impactful relationship between beautiful software architecture and software engineer productivity. To achieve their goal, the authors conducted a descriptive survey which contains sixteen questions for practitioners about their experiences and their perceptions. The participants filled in the survey online. Moreover, the authors conducted semi-structured interviews to identify how beauty in software architecture can be taught to the next generation. The participants of this study were eight and they are experienced architects. The interviews were conducted via video calls. The results of this study emphasize the positive influence on team dynamics and product quality. It stresses the importance of beauty, since ugly software architecture tends to frustrate and demotivate software engineers, thus decreasing productivity

3. Code Beauty Metrics

According to Gelernter "*Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defense against complexity*" (Gelernter, 1998). Code beauty refers to the qualities of code that make it not only functional but also elegant, readable, and maintainable. Also, as noted on the Software Engineering Stack Exchange[1], "*Most programmers will agree that beautiful code demonstrates a balance between clarity and transparency, elegance, efficiency and aesthetics*".

[1] https://softwareengineering.stackexchange.com/questions/207929/what-is-beautiful-code

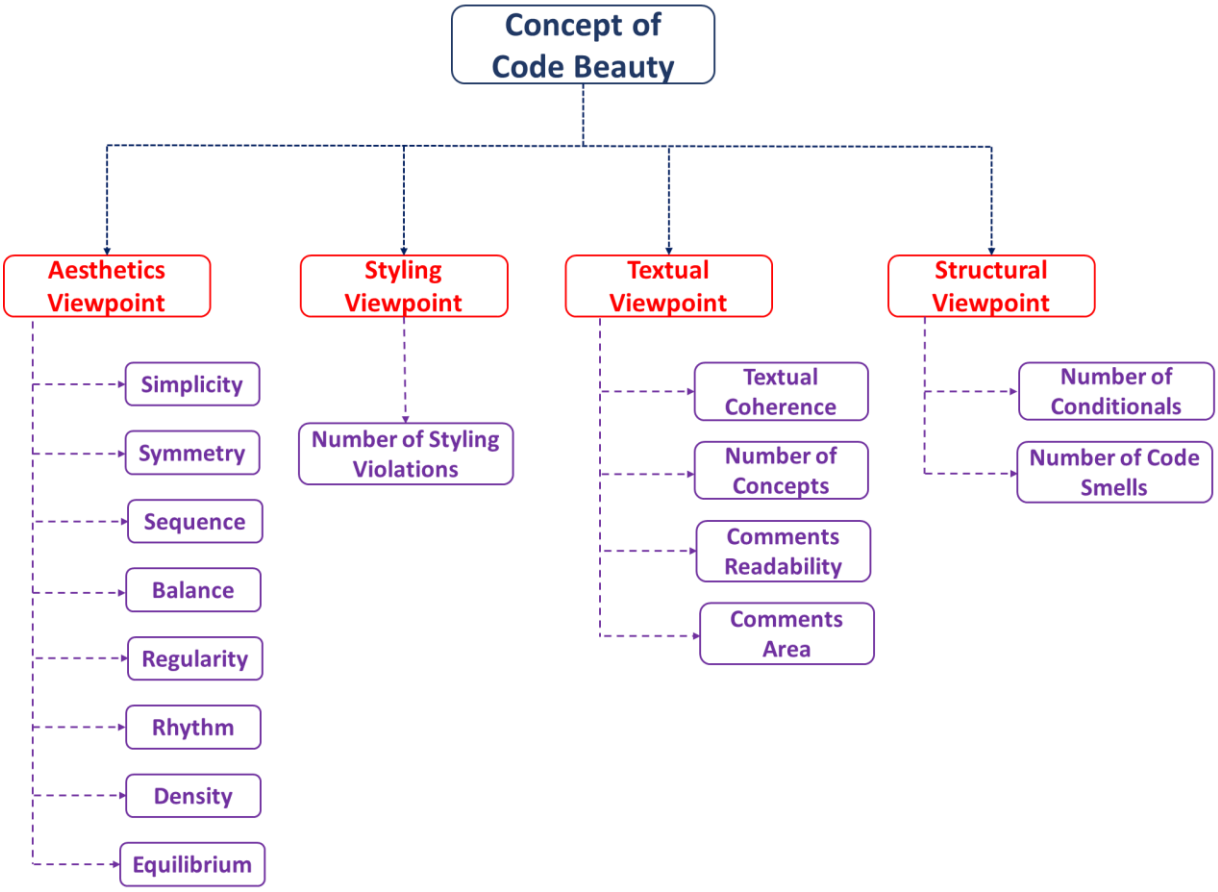


Figure 1. Code Beauty Scheme

In this study we propose a **Code Beauty Model (CBM)**—see Figure 1—that is organized hierarchically in three levels. The first level corresponds to the overall *concept of Code Beauty*. In the second level, code beauty is decomposed to four characteristics that provide a holistic view of code beauty, emphasizing both on technical and human-readable viewpoints of the code:

- *Aesthetics Viewpoint*. Code aesthetics refers to the visual and conceptual appeal of the code, focusing on how pleasing and intuitive it is to the reader. For example, the software engineer develops a code in a way that aligns naturally with its purpose, treating the code as an image or a painting (Dexter et al., 2011).
- *Styling Viewpoint*. Code styling focuses on the adherence of code to consistent and standardized practices in writing code. For example, the software engineer should follow writing guidelines (e.g., uniform formatting in terms of indentation and styling) (Oliveira et al., 2023).
- *Textual Viewpoint*. The textual structure of the code relates to the meaning that a code has when treated as natural language. For example, the use of appropriate comments clarifies intent or explains complex logic and improves readability without introducing redundancy; whereas the meaningful use of variable and method names enable the understanding and readability of the code, using well-established naming conventions (Cates et al., 2021).
- *Structural Viewpoint*. Considering that a software engineer cannot completely decouple the image from coding anti-patterns, code smells, and other structural inefficiencies, this viewpoint focuses on the structure of the code. For example, the selection of the appropriate design pattern (or avoidance of anti-patterns and code smells), the decision to split a large method into smaller, the proper use of classes, methods, and namespaces lead the code to be reusable and easier to understand (Connolly et al., 2023).

In the third level of CBM, we provide different metrics (e.g., simplicity, textual coherence, number of conditionals) for each characteristic. In Section 3.1 we present the proposed eight novel Code Aesthetic Metrics (CAM), since we have not been able to identify similar metrics in the literature. In Section 3.2 we present the used metrics in all other three viewpoints. These metrics have been identified in the literature, and therefore are presented together.

3.1 Code Aesthetic Metrics (CAM)

For the aesthetic evaluation of code, we consider the frame as a generic interface and evaluate it as such. An important resource for the aesthetic evaluation of interfaces has been published by Ngo et al. (2002). We note that in terms of contributions, our work goes in a different direction from Ngo et al. (2002), in the sense that his metrics targeted UI, whereas ours are tailored for source code assessment. Considering that we are interested in the overall aesthetic evaluation of code, we consider the complete width (***bframe***) and height (***hframe***) of a code file (frame) as our hypothetical interface borders. The frame of the code is divided into four quadrants by the intersection of the vertical and horizontal axes, ***UL*** (upper-left), ***UR*** (upper-right), ***LL*** (lower-left), and ***LR*** (lower-right). Another important aspect is the intersection of the vertical and horizontal axes, marked as a red circle and called ***geometric center***, which is determined by dividing the total width (bframe) and height (hframe) of the frame by two. In addition some other measures that play an important role in the calculation of our selected measures are: ***nvap*** that represents the number of vertical alignment points, calculated by counting the blank lines in the code; ***nhap*** that refers to the horizontal alignment points, determined by the number of distinct tab levels plus one to account for the zero-tab case; ***n*** that denotes the total number of blocks in the code, where blocks are defined as sections separated by either blank lines or tab indentations; ***b*** that represents the width of a line, defined as the total number of characters excluding the initial and intermediate tabs; ***h*** that refers to the height of the line; and ***l*** that represents the total number of lines in the code (see Figure 2). All selected metrics are presented briefly below; whereas a detailed presentation of their calculation is provided in Appendix A.

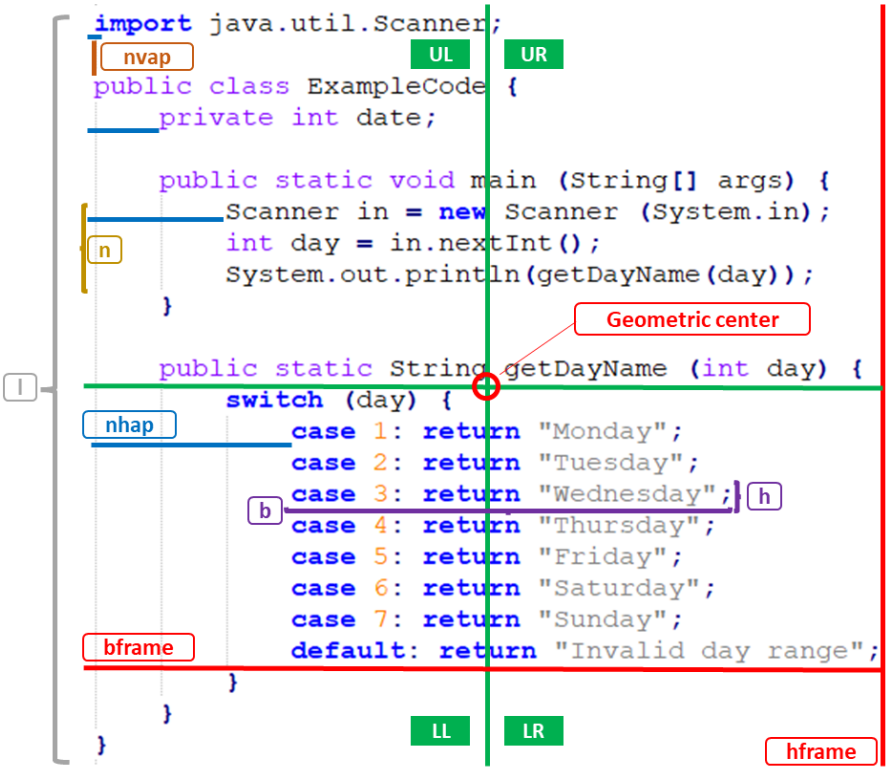


Figure 2. Code Aesthetics Metrics (CAM)

3.1.1 Simplicity Metric (SMM)

Simplicity in coding refers to the understandability of the code's layout based on its alignment points, non-blank lines and number of blocks (*n*). These alignment points can be either horizontal or vertical. Specifically, horizontal points (*nhap*) are the sum of distinct tabs in front of the non-blank lines plus 1, which refers to the "zero tab" case, while vertical points (*nvap*) are the sum of blank lines. Achieving simplicity involves minimizing both types of alignment points. The calculation of the metric has as a pre-condition the existence of one statement per line of code. In Figure 3, two examples of entities are presented with the left being simpler because of less blocks and alignment points (Ngo et al., 2002). Higher SMM values indicate greater simplicity, while lower values suggest more complex and potentially harder-to-read code. The detailed calculation of the SMM metric for the code of Figure 2 is presented in Appendix A.

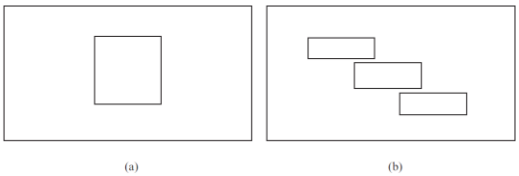


Figure 3. Left image is simpler because of fewer alignment points

3.1.2 Symmetry Metric (SYM)

Symmetry measures the extent of axial duplication. It measures how well characters of a code file exhibit horizontal, vertical, and radial symmetry. To achieve Symmetry, all units must be perfectly mirrored vertically, horizontally, or diagonally on all four quadrants (*UL*, *UR*, *LL*, and *LR*), considering the total x-distance of quadrant; total y-distance of quadrant; total height; total width; total angle as well as total distance all lines in quadrant from the geometric center of the frame. In Figure 4, we showcase two simple drawings, with the left exhibiting higher symmetry (Ngo et al., 2002). The detailed calculation of the SYM metric for the code of Figure 2 is presented in Appendix A.

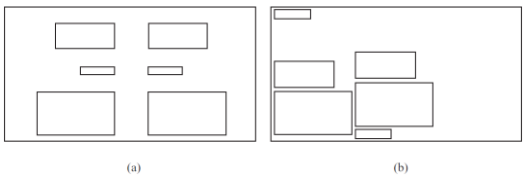


Figure 4. Left image is more symmetrical

3.1.3 Equilibrium Metric (EM)

Equilibrium measures the stability achieved when the center of the code aligns with the geometric center of the frame, ensuring a visually centered layout. Equilibrium can be defined along the X-Axis and along the Y-Axis. Regarding the X-Axis, it measures the alignment of lines of the code in the X direction (horizontal) relative to the center of the frame. Similarly, the equilibrium along the Y-axis, assesses the alignment of the lines of the code in the vertical direction (Y-axis).

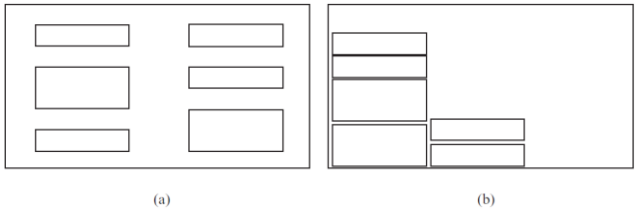


Figure 5. Left image exhibits a higher EM

In Figure 5 we see two images, wherein example (a) the mass is centered, whereas example (b) has a mass shifted towards the lower left corner (Ngo et al., 2002). A higher EM value suggests that the code's center of mass

aligns well with the center of the code, which contributes to a more visually balanced and aesthetically pleasing code layout. The detailed calculation of the EM metric for the code of Figure 2 is presented in Appendix A.

3.1.4 Rhythm Metric (RHM)

This measure evaluates whether the lines follow a distribution pattern and assesses the variety in both alignment points and line sizes. Unlike previous measures, good Rhythm value is achieved when there is diversity in the code layout as in a variety in both the alignment points and the lines sizes (Ngo et al., 2002). The basic aspects that influence RHM are Rhythm in the X-Axis, Y-Axis, and Covered Area. Rhythm in the X-Axis (RHMx) evaluates the variety in the x-distances between lines in different quadrants, while rhythm in the Y-Axis (RHMy) quantifies the variety in the y-distance between lines in different quadrants. In Figure 6 we see two examples of entities with "good" and "bad" rhythm. Example (a) contains diverse, yet structured and aligned entities, whereas example (b) has a very disorganized structure. Therefore, a higher RHM value suggests that the code exhibits a more diverse and aesthetically pleasing layout, contributing to improved code readability and visual appeal. The detailed calculation of the RHM metric for the code of Figure 2 is presented in Appendix A.

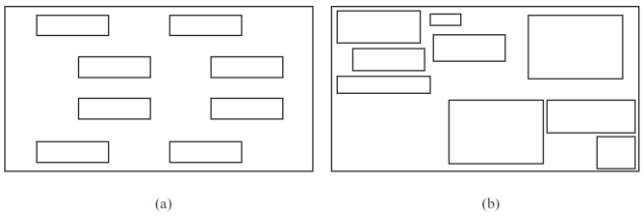


Figure 6. Left image has a higher rhythm

3.1.5 Regularity Metric (RM)

Regularity assesses the degree of consistency in alignment points and spacing among the distribution of lines. The basic aspects that influence RM are Alignment Regularity and Spacing Regularity. It aims to determine how well the lines align (Alignment Regularity) and are consistently spaced within the code (Spacing Regularity) (Ngo et al., 2002). More specifically, *nspacing* is the sum of the unique distances between the horizontal and between the vertical alignment points. In Figure 7 we see two examples of entities, where example: (a) contains consistently spaced entities vertically and horizontally, whereas example (b) does not (Ngo et al., 2002). The detailed calculation of the RM metric for the code of Figure 2 is presented in Appendix A.

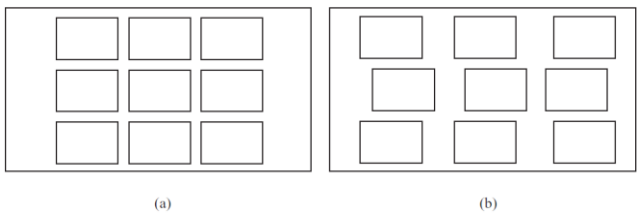


Figure 7. Left image is more regular

3.1.6 Sequence Metric (SQM)

Sequence is a metric that assesses the distribution of lines in the code and rates how well it follows the reading pattern commonly used in Western cultures (the eye, trained by reading, starts from the upper left and moves back and forth across the display to the lower right). To calculate the value of Sequence, we use the formula given by Ngo et al. (2002). The basic aspects that influence SQM are Quadrant Weighting and Vertical Alignment of lines within each quadrant (*UL*, *UR*, *LL* and *LR*). The former corresponds to the importance of each quadrant in reading, whereas the latter refers to the area occupied by the lines of code within each quadrant. In Figure 8, two examples of entities with "good" and "bad" sequence are presented (Ngo et al., 2002). Example (a) guides

the viewer according to the desired reading pattern, whereas example (b) has a more irregular pattern. Therefore, a higher SQM value suggests that the code distribution is closer to the expected reading pattern. The detailed calculation of the SQM metric for the code of Figure 2 is presented in Appendix A.

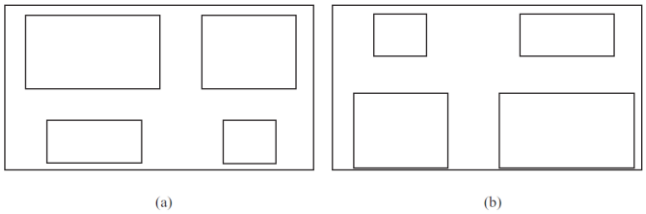


Figure 8. Left image follows a common reading sequence pattern

3.1.7 Density Metric (DM)

Density depicts the frame coverage with data, so in the case of code is the percentage of the frame covered with characters. Calculated by dividing the area covered by lines after removing any tabs or indentation, by the area occupied by the frame. This ratio provides an indication of how densely written the code is in relation to the available space provided by the code. In the context of code aesthetics, lower density values are often preferred, as they suggest a more visually pleasant and readable layout. Code that is too densely packed may be challenging to read and understand. In Figure 9 we see two graphic examples with the left being substantially less dense. (Ngo et al., 2002). The detailed calculation of the RHM metric for the code of Figure 2 is presented in Appendix A.

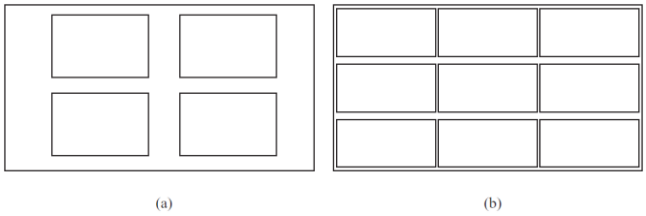


Figure 9. Left image has a higher density

3.1.8 Balance Metric (BM)

Balance is a metric related to the visual weight of code, particularly how the length and positioning of lines affect the perceived visual balance. Larger blocks of code appear “heavier” than smaller ones, thereby changing the perception of the viewer. To achieve Balance, all elements located above and below the center of the code on the y-axis must have the same weight. The same applies for the elements placed at both sides of the center on the x-axis. To calculate Balance, we consider the total weight of characters (non-blank characters); as well as the distance between the character and the center of the frame for the total number of lines on each side (*L*, *R*, *T*, and *B* stand for left, right, top, and bottom side). In Figure 10 we see two examples of entities with “good” and “bad” balance. Example (a) seems to be balanced, whereas example (b) is clearly imbalanced towards the right (Ngo et al., 2002). Therefore, a higher total BM value suggests that the code layout is visually balanced, with lines distributed in a way that creates an aesthetically pleasing and well-structured appearance. The detailed calculation of the BM metric for the code of Figure 2 is presented in Appendix A.

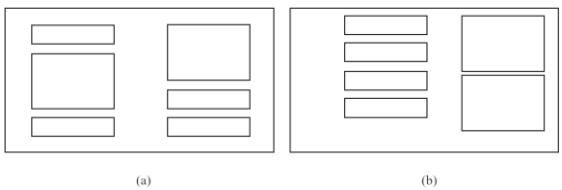


Figure 10. Left image is more balanced

3.2 Metrics for the Styling, Textual, and Structural Viewpoints

In this section we present the metrics that have been selected from the literature to capture the styling, textual, and structural viewpoints of the CBM. We note that although this list of metrics is not exhaustive (a long list of such metrics exists in the literature), we have selected those that appeared as more closely connected to the targeted characteristics. For the corresponding threat to validity emerging out of this selection, please check Section 6.3. The employed code beauty metrics are described below:

- **Number of Styling Violations** (Styling Viewpoint) is calculated as the number of styling issues that are identified by CheckStyle (Loriot et al., 2022). Checkstyle is a static code analysis tool that helps software engineers to write Java code that adheres to a coding standard.
- **Textual Coherence** (Textual Viewpoint) refers to the overlaps between terms used in pairs of syntactic blocks (Scalabrino et al., 2018). According to Scalabrino et al. (2018) this metrics is in the top-3 predictors of readability among the textual features.
- **Number of Concepts** (Textual Viewpoint) is calculated as the number of topics detected among the statements (Scalabrino et al., 2018). According to Scalabrino et al. (2018) Number of Concepts is among the top-3 predictors of readability among the textual features.
- **Comments Readability** (Textual Viewpoint) is calculated as the Flesch-Kincaid reading-ease score (FRES) of the comments linked to methods (Scalabrino et al., 2018). According to Scalabrino et al. (2018) Comments Readability is the best predictor of readability among the textual features.
- **Comments Area** (Textual Viewpoint) refers to the ratio of characters that are commented compared to the total number of characters in the code (Scalabrino et al., 2018). The number of comments in the code is expected to be positively related to code readability and understandability (Buse and Weimer, 2010).
- **Number of Code Smells** (Structural Viewpoint) is calculated as the number of code smells identified by SonarQube (Letouzey, 2012). SonarQube performs an analysis and generates a report to ensure reliability.
- **Number of Conditionals** (Structural Viewpoint) refers to the number of conditional (e.g., if, switch) and loop (e.g., for, while) statements that exist in the code (Buse and Weimer, 2010). In software engineering literature conditional statements have been related to the cognitive complexity of the code, hindering code readability (Campbell, 2018).

4. Empirical Validation

4.1 Objectives & Research Questions

To serve the high-level goal of this study ("*Explore if code beauty can be correlated to the quality of the code, and more specifically to software maintainability"*), we have performed two empirical studies, targeting the 3 sub-goals explained in the introduction. Each subgoal is in this section transformed to a research question (RQ):

[RQ1] Is code beauty a concept that is uniformly perceived by different individuals?

[RQ2] Are CBM metrics correlated to perceived code beauty?

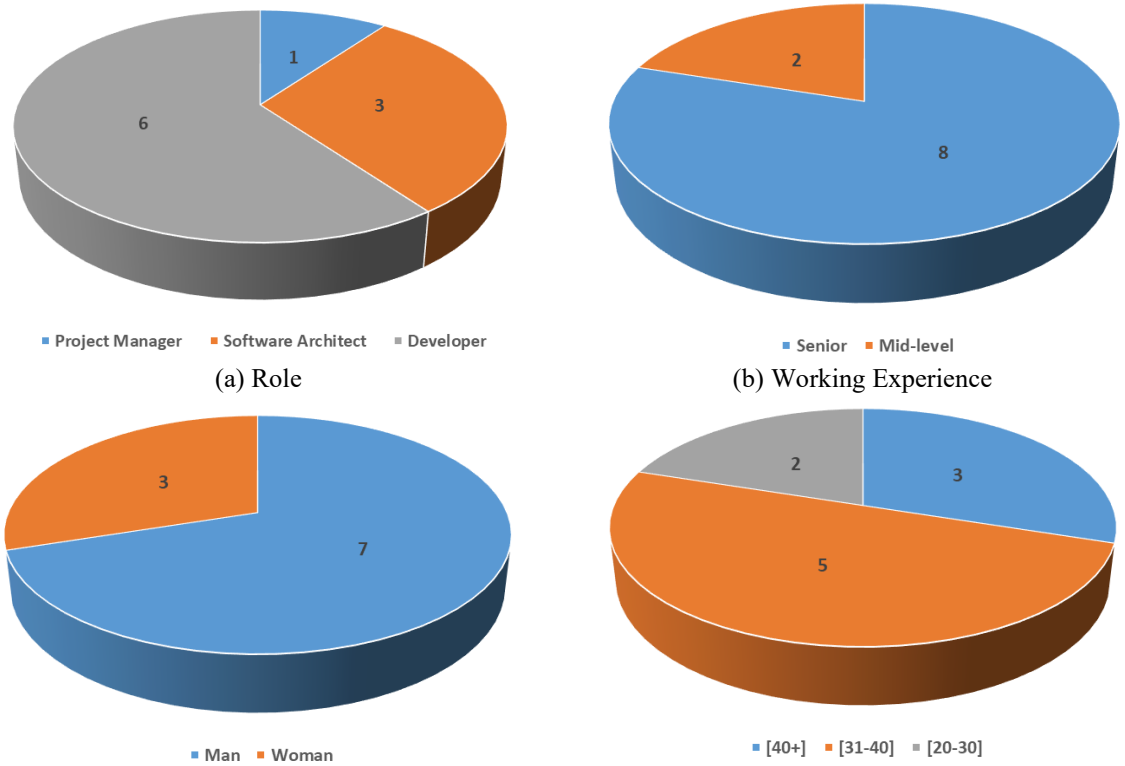
[RQ3] To what extent does code beauty correlate to software maintainability?

RQ1 and RQ2 are answered through a ***study with human participants*** (i.e., senior software engineers). The aim is to explore the extent to which software engineers can assess code beauty in the same way, providing an assessment on the subjectivity of the code beauty matter. Additionally, we explore the extent to which the CBM metrics are accurate, i.e., if they capture the code beauty phenomenon. ***RQ3*** is answered through an ***analysis of existing software artifact***. Given the two different setups, we report the two studies separately in the upcoming sections. A detailed replication package has been developed and is available online[2].

[2] https://users.uom.gr/~a.ampatzoglou/aux_material/code_beauty_replication.zip

4.2 Case Selection and Units of Analysis

Study with Human Participants: As subjects for this study, we have recruited 10 SE professional. Some demographics, of the participants are presented in Figure 11 (the experience is measured in years).



(a) Role (b) Working Experience

(c) Gender (d) Age

Figure 11. Study Demographics

Artifact Analysis Study: The subjects for this study are open-source software projects, and the units of analysis are classes. The reporting is performed cumulatively for the complete dataset, and we do not separate per software project. The decision to not explore the project parameter does not influence the validity of the study in the sense that our work is focusing on specific files, and the studied relation is not expected to change, due to organizational aspects of the project. We note that this study solely depends on source code analysis and no humans are involved. Our dataset consists of more than 27,000 classes, which correspond to the complete set of classes from 11 Java open-source projects. The selected projects along with some basic descriptive statistics are presented in Table 1. The selection of projects has been reused from the work of Nikolaidis et al. (2023), following three criteria: i.e., the projects are written in Java to enable our static analysis, they are having active development to ensure that they are up-to-date subjects, and they are having substantial history that uses a structured committing process to ensure that they are mature projects.

Table 1. Selected Projects

Name	Lines of Code	Number of Classes
Antlr4	44,613	421
Conductor	53,488	507
DD-trace-java	175,482	2,686
Dolphin Scheduler	107,772	1,873
Druid	981,231	7,140
Dubbo	200,404	3,407

Name	Lines of Code	Number of Classes
Incubator Sea Tunnel	92,462	1,797
Pulsar	527,708	3,899
Rocket MQ	169,725	1,698
Sky-walking	75,137	1,722
Stream-pipes	60,712	2,144

4.3 Data Collection

The data collection process can be split into three parts, the first one for recording the Perceived Code Beauty (PCB) to be used for answering RQ_1 and RQ_2; the second one for calculating CBM metrics; and the third for the collection of the maintainability predictors. CBM metrics have been calculated in two contexts. For answering RQ_1 and RQ_2 they have been calculated for the 20 sample code chunks of the study with human participants, whereas for answering RQ_3 they have been calculated for the open-source projects of Table 1. The 20 sample code chunks have been developed by the first author of the publication. All code chunks are delivering the same functionality to filter out the confounding factor of functionality in the "*first-impression*" evaluation of the software engineers. Thus, the code chunks are functionally equivalent, but are using different logic to deliver it, leading to source codes of different shapes—also differing in terms of CBM metrics scores. More specifically, in our final dataset, we have attempted to include as many combinations of HIGH / MEDIUM / LOW scores for each CBM metrics, as possible. The set of code chunks along with the presentation of the CAM calculation are presented in Appendix B.

Assessment of Perceived Code Beauty (PCB): To assess code beauty each one of the 10 participants have been given the code chunks and have been asked to rank them from 1 to 20 (not allowing ties), based on their perceived code beauty, with 1 being the most aesthetically pleasing and 20 being the least. From this process, we retrieved a dataset for which the rows corresponded to individual raters, whereas the columns to the rank of the specific code chunk in terms of perceived beauty by the corresponding rater. During the ranking phase, we have not allowed for equal ranking in the beauty of two different code chunks, since in this data collection step we are interested in prioritization. As a metric of PCB, we have computed the mean score of ranking assessments for PCB (MR_{PCB}) for each code chunk, by all participants. We note that PCB and MR_{PCB} are inversely proportional, i.e., a code chunk with low MR_{PCB} is the most pleasing aesthetically for most participants.

Code Beauty Model Metrics: The second part of data collection aimed at calculating the metric scores of CBM metrics for the classes of the dataset. To collect the metrics, we have developed a tool for CAM assessment, and we have reused SonarQube, CheckStyle, and RSM. The CAM assessment tool[3] is available online and has been tested in various settings: (a) small-scale projects for which CAM were calculated, manually, and contrasted to the automatically extracted scores; (b) large-scale projects for identifying abnormal (outside metric range) score; and (c) checking common static analysis mistakes that we have catalogued over the years in other source code parsers. All CBM metrics have been calculated in the last version of the selected projects.

Maintainability Predictors: According to Riaz et al. (2009) the major quality characteristics that can be used as maintainability proxies are coupling, complexity, cohesion. To quantify these quality characteristics, we have selected the following metrics:

- Cyclomatic Complexity (CC): This metric measures the complexity of a program by counting the number of linearly independent paths through the code. CC is considered as the state-of-the-art complexity metric because it is well-established and well-tested in terms of its relation to maintainability. CC, when compared to other complexity metrics, considers the internals structure of a method, enabling the capture of the actual complexity of the class (McCabe, 1976).

[3] https://github.com/teomaik/Code_Beauty_Calculator.git

- Lack of Cohesion of Methods (LCOM): LCOM measures the lack of cohesion among methods within a class. It quantifies the number of method pairs that do not share any instance variables. This metric has been selected since: (a) high cohesion is one of the most important principles of object-orientation, and (ii) lack of cohesion directly implies the existence of the large class "bad smells", which urges for the application of refactoring (Chidamber and Kemerer, 1994).
- Message Passing Coupling (MPC): MPC measures the number of distinct methods called from a class. MPC has been selected since it is the only coupling metric that captures both coupling volume (number of relationships) and coupling intensity (how closely connected the two classes are). An additional characteristic of MPC is that it counts coupling intensity using the discrete count function, and therefore is not biased from the number of times one method is being called (Li and Henry, 1993).
- Lines of Code (LoC): This is a simple metric that counts the number of lines in the source code. It is a basic measure of the size of the codebase. LoC can give an indication of the scale of a class or project. LoC is used in almost all maintainability studies and is accredited as a top predictor of maintenance load, which is a basic component of TD Interest calculation (Chidamber and Kemerer, 1994).

The selected metrics have been indicated by previous as the optimal maintainability predictors: Riaz et al. (2009) have performed a quality assessment of maintainability models, through a quantitative checklist, to identify studies that provide reliable evidence on the link between metrics and maintainability. Among the studies with the highest scores were those of van Koten and Gray (2006) and Zhou and Leung (2007). Both studies have been based on two metric suites proposed by Li and Henry (1993) and Chidamber and Kemerer (1994). The employed suites contain metrics that can be calculated at the source-code level, and can be used to assess well-known quality properties, such as inheritance, coupling, cohesion, complexity, and size. To calculate the scores for the aforementioned metrics, we used Metrics Calculator[4], a well-tested and stable tool for calculating quality metrics for Java code.

4.4 Data Analysis

To answer the posed RQs, we performed appropriate multivariate statistical methods and ML modelling techniques for accomplishing the objectives that are described in Section 4.1. More specifically, for RQ_1, we made use of exploratory visualization techniques for examining how SE participants perceived code beauty, based on their ranking assessments on the set of the inspected code chunks. Next, the *Kendall's W coefficient of concordance* (Kendall, 1948) was evaluated for the total number of the participants to infer about the degree to which multiple raters rank the beauty of the set of code chunks consistently.

Concerning RQ_2, we developed a *Random Forest* (RF) model that was built between MR_{PCB} (response) and the set of CBM metrics (features). Regarding the selection of the significant features to be inserted into the building of the RF model, we made use of a well-known wrapper algorithm, known as the Boruta algorithm (Kursa and Rudnicki, 2010). In brief, the algorithm evaluates the importance of each feature by comparing it to randomly shuffled data (shadow features) combined with the original values of features. This iterative process results in the characterization of predictors as informative or non-informative features through formal hypothesis testing procedures. In addition, to gain insights on the subset of CBM metrics: (a) that have the most impact on PCB; and (b) how their values affect MR_{PCB}, by applying a well-established *eXplainable Artificial Intelligence* (XAI) method, namely: *SHapley Additive exPlanations* (SHAP) analysis (Lundberg and Lee, 2017).

Finally, for RQ_3, we conducted correlation analysis, focusing on association rather than causation, between the set of CBM metrics that are related to PCB (based on RQ_2) and maintainability metrics using the *Spearman's correlation coefficients* (Winter et al., 2016).

[4] https://github.com/dimizisis/metrics_calculator.git

5. Findings and Answers to the Research Questions

In this section, we present the results of this study, organized by RQ. For simplicity, we have preferred to present the results in a raw way in this section and discuss them (interpretations and implications) in Section 6.

5.1 Objectivity of Perceived Code Beauty (RQ_1)

To answer RQ_1, we have explored the ranking assessments distributions of each code chunk among individual software developers (see Figure 12). The investigation of the distributions suggests that regarding the beauty of the examined code chunks, there is noted a moderate level of agreement among software engineers. For example, the cases of "Code Chunk 4" and "Code Chunk 3" highlight that software developers assessed as less "*beautiful*" (i.e. higher MR_{PCB} denoted by red colour) these specific code chunks, as all the respondents (100%) gave a ranking that is higher or equal to 11 $(min = 1\ (most\ aesthetically\ pleasing), max = 20)$. In contrast, a consensus among respondents is noted regarding the "beauty" for "Code Chunk 6", "Code Chunk 13" and "Code Chunk 20". The findings from the visual inspection of the distributions were validated by the results of the inter-rater agreement analysis through the computation of the Kendall's W concordance coefficient. Indeed, the analysis revealed a statistically significant and moderate agreement among the total set of the participants ($W = 0.485, p < 0.001$).

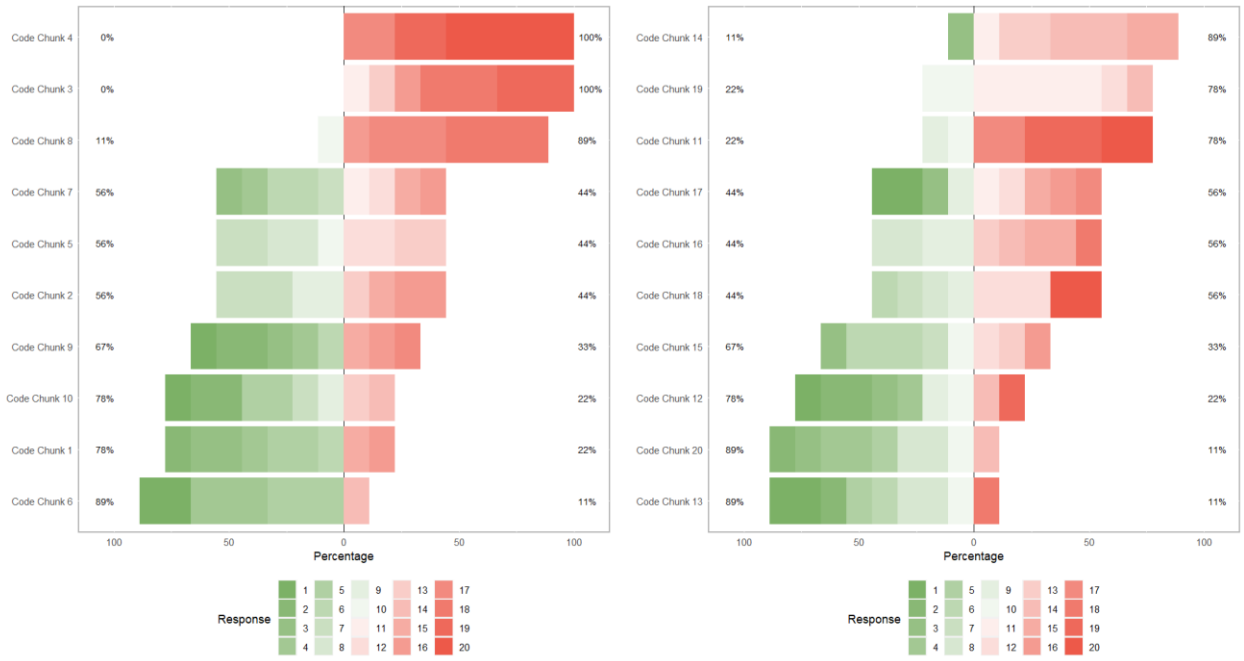


Figure 12. Distribution of responses for PCB

This finding suggests that code beauty is something that cannot be uniformly perceived by software engineers, buy some common perception of what code beauty is, probably exists with deviations.

5.2 Assessing Perceived Code Beauty from Code Aesthetics and Other Beauty Metrics (RQ_2)

RQ_2 focuses on the investigation of whether a set of CBM metrics can serve as suitable proxies for quantifying PCB. The analysis was based on CBM metrics values for the total set of 20 code chunks for which descriptive statistics were evaluated and summarized in Table 2. To investigate if there is a subset of CBM metric that was able to capture perceived beauty, an RF model was, initially, built between MR_{PCB} and the subset of features (CBM metrics) that were deemed as important from the application of the Boruta algorithm, whereas SHAP analysis was adopted for interpretability purposes. During the tuning phase of the model, a grid search with repeated cross-validation was performed to identify the optimal hyper-parameters including the number of

estimators (i.e., the number of trees in the forest), maximum depth of the trees, the minimum number of samples required to split an internal node, the minimum number of samples required to be at a leaf node, the maximum number of features considered for splitting, and whether to bootstrap samples when building trees. This comprehensive approach ensured that the model's performance was maximized by selecting the best combination of hyperparameters. Regarding the informative features identified by the execution of the Boruta algorithm, we have identified five predictors ("*Number of Conditionals*", "*Regularity*", "*Comments Area*", "*Number of Code Smells*" and "*Simplicity*") out of a total set of fifteen candidate predictors (see Table 2) to be statistically significant ($p < 0.05$) for insertion into the model building phase of the RF model. The fitted model achieved a noteworthy performance in terms of Mean Absolute Error ($MAE = 0.73$), Mean Squared Error ($MSE = 0.82$) and R-squared ($R^2 = 0.95$) indicators suggesting that a relatively high proportion of variability in PCB can be explained by the examined set of CBM metrics.

Table 2. Descriptive statistics of CBM metrics for the total set of 20 Code Chunks

CBM Metrics	**Statistical Measure**	**Overall (*N*=20)**
Simplicity Metric (SMM)	*Mean* (*SD*)	0.121 (0.021)
	Median (*min*, *max*)	0.125 (0.083, 0.176)
Symmetry Metric (SYM)	*Mean* (*SD*)	0.823 (0.075)
	Median (*min*, *max*)	0.830 (0.693, 0.925)
Equilibrium Metric (EM)	*Mean* (*SD*)	0.557 (0.039)
	Median (*min*, *max*)	0.561 (0.486, 0.617)
Rhythm Metric (RHM)	*Mean* (*SD*)	0.815 (0.080)
	Median (*min*, *max*)	0.807 (0.691, 0.944)
Regularity Metric (RM)	*Mean* (*SD*)	0.862 (0.025)
	Median (*min*, *max*)	0.858 (0.833, 0.920)
Sequence Metric (SQM)	0.5 (*N*, %)	8 (40%)
	0.75 (*N*, %)	12 (60%)
Density Metric (DM)	*Mean* (*SD*)	0.531 (0.091)
	Median (*min*, *max*)	0.526 (0.369, 0.698)
Balance Metric (BM)	*Mean* (*SD*)	0.559 (0.137)
	Median (*min*, *max*)	0.500 (0.403, 0.897)
Number of Styling Violations	*Mean* (*SD*)	7.750 (1.118)
	Median (*min*, *max*)	8.000 (6.000, 10.000)
Number of Code Smells	*Mean* (*SD*)	3.400 (2.644)
	Median (*min*, *max*)	2.000 (2.000, 9.000)
Textual Coherence	*Mean* (*SD*)	0.941 (0.071)
	Median (*min*, *max*)	0.969 (0.794, 1.000)
Number of Concepts	*Mean* (*SD*)	4.723 (0.672)
	Median (*min*, *max*)	4.464 (3.942, 6.565)
Comments Readability	*Mean* (*SD*)	0.724 (0.095)
	Median (*min*, *max*)	0.726 (0.541, 0.838)
Comments Area	*Mean* (*SD*)	0.087 (0.019)
	Median (*min*, *max*)	0.088 (0.041, 0.121)
Number of Conditionals	*Mean* (*SD*)	3.500 (2.085)
	Median (*min*, *max*)	3.000 (1.000, 8.000)

Next, the focus has shifted to the examination of the findings that were extracted from the SHAP analysis on the fitted RF model with an emphasis on the identification of CBM metrics that highly affected MR_{PCB} (and subsequently PCB). Figure 13 illustrates how each metric contributed to MR_{PCB} fitted values in decreasing order of importance as quantified by the mean absolute SHAP values. The feature with the highest impact is "*Regularity*", with a mean absolute SHAP ($mean|SHAP|$) value of 0.870, followed by "*Comments Area*" ($mean|SHAP| = 0.803$), "*Number of Code Smells*" ($mean|SHAP| = 0.783$) and "*Number of Conditionals*" ($mean|SHAP| = 0.434$). In contrast, "*Simplicity*" seems to have a rather small effect on MR_{PCB} with a value of $mean|SHAP| = 0.265$. Based on the findings related to the importance of each metric on MR_{PCB} (see Figure 13), we decided to further investigate the underlying relationships between the actual values of the informative features and code beauty. To achieve this objective, we made use of the global SHAP force plots that constitute a straightforward visualization XAI technique for the examination of the impact of the estimated Shapley values on the response value.

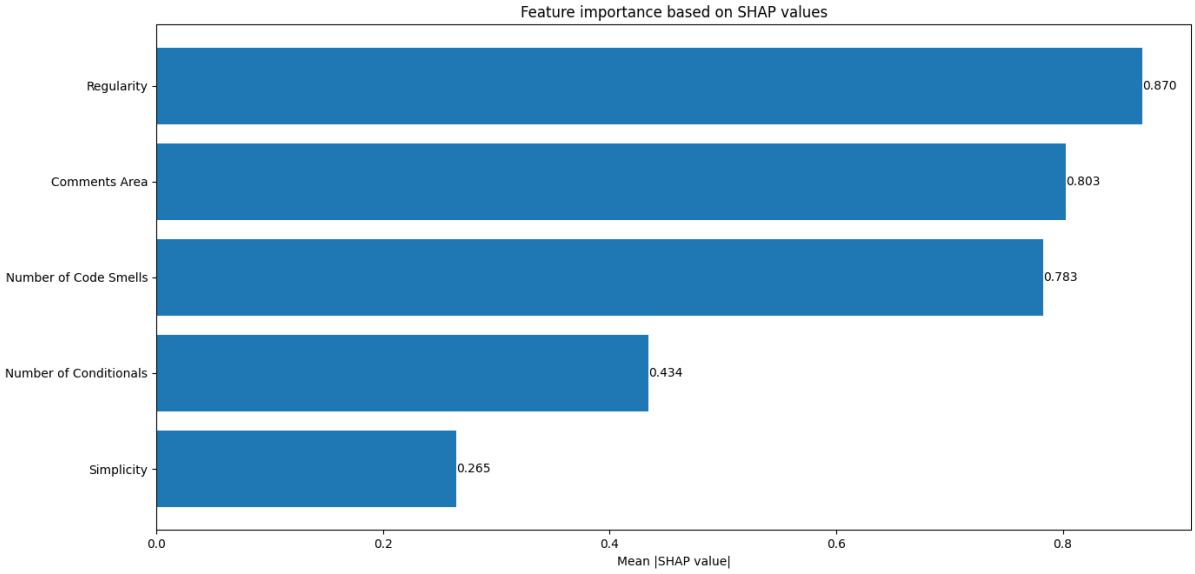


Figure 13. SHAP feature importance plot

Figures 14-16 display the global SHAP force plot for three illustrative metrics that influenced code beauty in opposite directions. In these plots, the x-axis and the y-axis represent the values of the examined feature, and the values of the response variable (MR_{PCB}) derived from the fitted RF model, respectively. Moreover, the negative and positive Shapley values are shaded with blue- and red- colored regions, respectively, to facilitate the examination of the effect of the feature on the estimated response values. For example, the inspection of Figure 14 reveals that higher values of "*Number of Conditionals*" are associated with high mean scores of ranking assessments and thus, lower perceived beauty. In addition, the "*Number of Conditionals*" value close to 4 can be considered as the "*break point*", at which the model switches from lower (negative Shapley values) to higher (positive Shapley values) mean ranking scores of perceived code beauty. In simple words, "*Number of Conditionals*" values above 4 contribute to less aesthetically pleasing code. This trend also holds for Regularity and Number of Code Smells (see Figure 15). In contrast, Figure 16 shows that as "*Comments Area*" increases, the mean scores of ranking assessments tend to decrease, which practically means that the code is perceived as more aesthetically pleasing, while the value of "*Comments Area*" close to 0.08 can be considered as the breaking point.

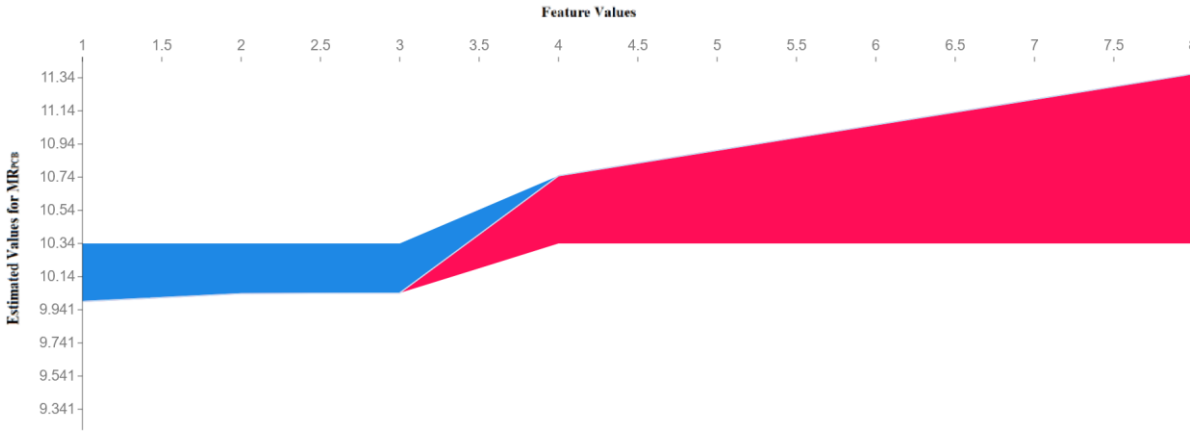


Figure 14. Global SHAP force plot for Number of Conditionals

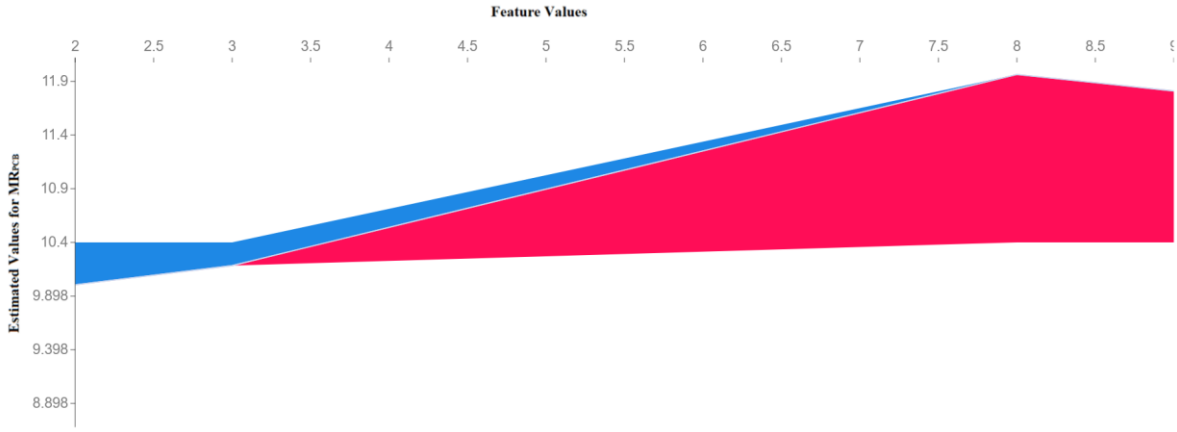


Figure 15. Global SHAP force plot for Number of Sonar Issues

Figure 16. Global SHAP force plot for Comments Area

Summarizing, the Boruta algorithm signified that a set of five Code Beauty Model metrics can be considered as informative and effective proxies for quantifying perceived code beauty. Additionally, SHAP analysis revealed that Simplicity (Aesthetics Viewpoint) and Number of Conditionals (Structural Viewpoint) have a small impact on PCB as they exhibited low SHAP values, that practically means, their contributions to the model are relatively small. In contrast, another Structural (Number of Code Smells) and one Textual Viewpoint (Comments Area) metrics along with Regularity (Aesthetics Viewpoint) are significantly more influential than the other features.

5.3 Relation of Code Beauty Model Metrics and Maintainability Predictors (RQ_3)

RQ_3 is dedicated to the exploration of the nature and strength of the correlation between CBM metrics and Maintainability Predictors. In this regard, a total set of more than 21,000 classes from 11 Java open-source projects were analyzed. In Table 3, we present the descriptive analytics for CBM metrics and the Maintainability Predictors.

Table 3. Descriptive statistics for each CBM metrics and Maintainability Predictors

Suite	**Metric**	**Statistical Measure**	**Overall (*N*=21,137)**
CBM Metrics	Simplicity (SMM)	*Mean* (*SD*)	0.092 (0.076)
		Median (*Q1*, *Q3*)	0.071 (0.039, 0.125)
	Regularity (RM)	*Mean* (*SD*)	0.823 (0.068)
		Median (*Q1*, *Q3*)	0.831 (0.777, 0.874)
	Number of Conditionals	*Mean* (*SD*)	12.448 (68.900)
		Median (*Q1*, *Q3*)	3.000 (1.000, 10.000)
	Comments Area	*Mean* (*SD*)	0.320 (0.256)
		Median (*Q1*, *Q3*)	0.272 (0.102, 0.511)
	Number of Code Smells	*Mean* (*SD*)	1.998 (5.517)
		Median (*Q1*, *Q3*)	0.000 (0.000, 2.000)
Maintainability Predictors	Cyclomatic Complexity (CC)	*Mean* (*SD*)	1.305 (1.775)
		Median (*Q1*, *Q3*)	1.000 (0.500, 1.500)
	Lack of Cohesion of Methods (LCOM)	*Mean* (*SD*)	45.354 (680.224)
		Median (*Q1*, *Q3*)	0.000 (0.000, 6.000)
	Message Passing Coupling (MPC)	*Mean* (*SD*)	15.502 (34.495)
		Median (*Q1*, *Q3*)	5.000 (0.000, 17.000)
	Lines of Code (LoC)	*Mean* (*SD*)	62.824 (141.151)
		Median (*Q1*, *Q3*)	23.000 (6.000, 66.000)

To provide an answer to RQ_3, we evaluated the Spearman's rho correlation coefficient for each pair of the set of CBM metrics and code quality metrics. Table 4 summarizes the descriptive statistics for the total number of the pair-wise comparisons between the set of important CBM metrics and Maintainability Predictors, in which all pair-wise comparisons resulted into a statistically correlation ($p < 0.001$). Rows represent maintainability predictors and columns CBMs.

Table 4. Descriptive statistics of correlation coefficients for each pair of CBM and Maintainability Metrics

	Measure	**Regularity**	**Simplicity**	**Number of Conditionals**	**Comments Area**	**Number of Code Smells**
CC	*Mean* (*SD*)	0.712 (0.050)	-0.720 (0.060)	0.834 (0.065)	-0.513 (0.299)	0.303 (0.125)
	Median (*min*, *max*)	0.740 (0.630, 0.759)	-0.742 (-0.792, -0.588)	0.853 (0.702, 0.898)	-0.661 (-0.748, 0.174)	0.277 (0.097, 0.522)
LCOM	*Mean* (*SD*)	0.563 (0.158)	-0.606 (0.171)	0.644 (0.156)	-0.357 (0.230)	0.262 (0.162)
	Median (*min*, *max*)	0.552 (0.332, 0.807)	-0.602 (-0.854, -0.340)	0.627 (0.435, 0.874)	-0.435 (-0.583, 0.184)	0.221 (0.073, 0.545)

	Measure	Regularity	Simplicity	Number of Conditionals	Comments Area	Number of Code Smells
LoC	*Mean* (*SD*)	0.784 (0.071)	-0.917 (0.032)	0.879 (0.034)	-0.732 (0.384)	0.470 (0.149)
	Median (*min*, *max*)	0.774 (0.698, 0.887)	-0.923 (-0.951, -0.845)	0.886 (0.809, 0.922)	-0.888 (-0.948, 0.218)	0.473 (0.246, 0.692)
MPC	*Mean* (*SD*)	0.624 (0.121)	-0.750 (0.061)	0.729 (0.098)	-0.635 (0.357)	0.419 (0.153)
	Median (*min*, *max*)	0.640 (0.430, 0.788)	-0.773 (-0.843, -0.658)	0.748 (0.555, 0.853)	-0.797 (-0.853, 0.239)	0.422 (0.169, 0.653)

To extract meaningful conclusions concerning the distributions of the pair-wise comparisons between the CBM and maintainability metrics, we constructed the corresponding boxplots (Figure 16). A first interesting remark concerns the direction and strength of the relationship between CBM and maintainability metrics. "*Simplicity*" presents a statistically significant negative correlation with all maintainability metrics ranging from moderate ($min = -0.534$) to very strong ($max = -0.951$) for the cases of LCOM and LoC, respectively. Additionally, the interquartile range of the boxplots suggest a low amount of variability in the extracted findings apart from LCOM. This negative trend is also observed for most projects concerning the comparison between "*Comments Area*" and the maintainability metrics. The remaining CBM metrics are positively correlated with all maintainability metrics. For example, "*Regularity*" is positively correlated with all maintainability metrics that ranges from low ($min = 0.332$) for the case of LCOM to very strong ($max = 0.887$). Finally, although statistically significant correlations are observed between each pair of CBM metrics and "*Number of Code Smells*", the strength of these associations is, generally, weaker compared to other CBM metrics, a result that is reflected through the examination of the distribution ranges.

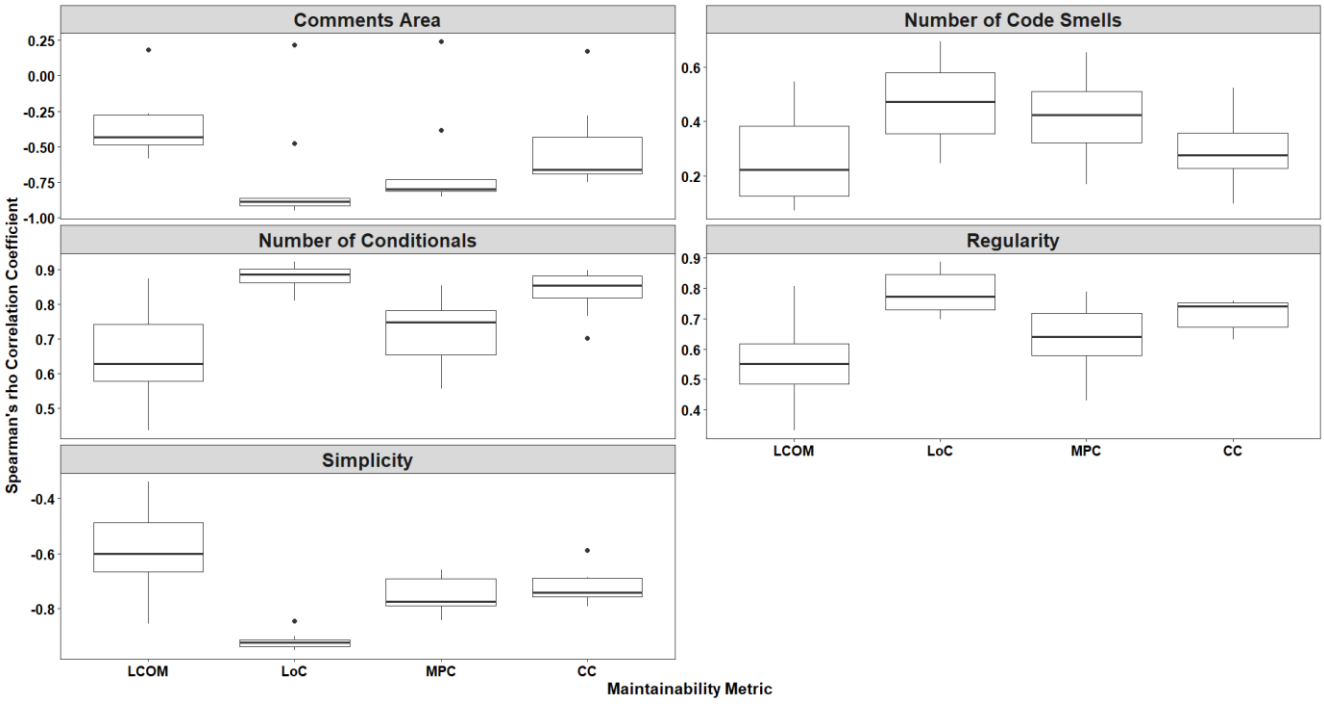


Figure 16. Distributions of Spearman's rho correlation coefficient for Maintainability and CBM Metrics

Regarding cohesion (LCOM) the stronger relation appear with CAMs (Simplicity and Regularity). Size (LoC) appears to be highly correlated with every CBM metric, with exceptional high correlation (with very limited variability) with Simplicity and Comments Area. Regarding code complexity (CC), we can observe an almost perfect correlation with Number of Conditionals and a strong one with CAMs. Finally, regarding coupling (MPC) we can observe a strong relation with CAMs and Comments Area. "Reading" the correlations in a vice-verse way, we can observe that all CBM metrics are stronger correlated to LoC. Neglecting the obvious relation to project size, Comments Area, Number of Code Smells, and Simplicity are highly correlated with MPC, and Regularity with CC.

6. Discussion

6.1 Interpretation of Results

In this section, we summarize the most important findings and interpret them. Upon analyzing the inter-rater agreement results, it becomes evident that there was a moderate and statistically significant agreement across all software engineers. It highlights that software engineers tend to share a cohesive understanding of code beauty, albeit with some deviations possibly influenced by personal preferences. The overall agreement on what constitutes "*beautiful code*" among software engineers implies that training and experience in software engineering leads to a quite consistent criterion for good code.

Next, we performed an aesthetic metrics analysis on 20 survey code files. Using MR_{PCB}, as the target, and the calculated CBM metrics, as features, we trained a Random Forest regression model. We then performed a SHAP analysis to extract the feature importance of each CBM metrics. By using this XAI approach, we aimed not only at identifying the magnitude of CBM metrics influence, but also the relationship the CBM metrics value has to the PCB. The findings reveal that software engineers value "*Number of Conditionals*", "*Regularity*", "*Comments Area*", "*Number of Code Smells*", and "*Simplicity*", when assessing the beauty of a code chunk. Regarding beauty characteristics, the results suggested that the Aesthetics Viewpoint and the Structural Viewpoint are represented with two metrics in the most important features for perceived code beauty, followed by the Textual Viewpoint. The Styling Viewpoint is not represented; however: (a) in this study there was only one generic metric for this characteristic; and (b) this characteristic is indirectly captured by other CBM metrics (e.g., the number of indentations used is considered in the calculations of CAMs and is proxied by the "*Number of Conditionals*"). The result on CAMs comes as no surprise, as simple code (captured by "Simplicity") has been advocated by numerous authors in software engineering literature, as one of the primary rules for clean, maintainable and understandable software (Martin, 2008). Similarly, a code that is "Regular" is expected to lack indentions (and probably styling), leading to intuitively being perceived as less aesthetically pleasing to software engineers (Oliveira et al., 2023). In terms of structure, the fact that a code with more comments ("*Comments Area*") is regarded as more "beautiful" is also intuitive, in the sense that: (a) the colored lines of comments (in most IDEs) are easily spotted in the first-impression of the code; (b) the belief that comments is a way to safeguard understandability is well accepted among software engineers (Cates et al., 2021), making software engineers as positively engaged with a code chunk with comments directly. The second line of interpretation suggests that most engineers consider comments as part of beautiful code, and not as a sign of poor code that "needs the comment" to be interpreted. This line of interpretation also applies to other metrics of the Textual Viewpoint (e.g., "*Comments Readability*" and "*Textual Coherence*"), which are also acknowledged as important for readability (Cates et al., 2021), but cannot be captured by the "*first-look*" of the code. Finally, the structural assessment of code also seems to take place, at least to some extent, in the first minute look, in the sense that both "*Number of Conditional*" and "*Number of Code Smells*" are related to PCB. If the "Number of Conditionals" (especially on well-styled or beautified code) is excessive, it can be easily spotted in the "*first-look*" impression and directly alerts the reader of a complex code chunk (McCabe, 1976). Similarly, the fact that

the "*Number of Code Smell*" is less important stems from the fact that several code smells (e.g., God class (Fowler et al., 1999), long methods (Fowler et al., 1999), magic numbers (Campbell, 2013), etc.) can be spotted very quickly, whereas others (e.g., feature envy (Fowler et al., 1999) or replace conditionals with polymorphism (Fowler et al., 1999)) require a timely code inspection.

Lastly, to evaluate the aesthetic metrics connection with maintainability predictors, we performed a correlation analysis on eleven Java open-source projects. The CBM metrics we examined are the five ones which are valued the most by software engineers when assessing code beauty, namely "*Number of Conditionals*", "*Regularity*", "*Comments Area*", and "*Number of Code Smells*" and consider the "*Simplicity*". The interpretation of the results of this endeavor can be performed in two ways (i.e., from maintainability predictors to CBM metrics and vice-verse). The first finding is that all CBM metrics are correlated to code size (LoC), which is supported (either directly or indirectly) by the definition of CBM metrics and the quality laws of evolution (Herraiz et al., 2013). In this respect, we can underline that the correlations with size were almost perfect for non-structural CBMs, but moderate to strong for the Structural Viewpoint metrics ("*Number of Conditionals*" and "*Number of Code Smells*"), suggesting that at population level there are cases where a small code chunk might concentrate structural mishaps, and larger codes of good quality. An interesting finding is that code cohesion (LCOM) is better captured in the first minute inspection by the Aesthetics Viewpoints metrics. This finding cannot be explained by the definition of the CAMs and has been raised as a rather unexpected result from the empirical analysis. The relation between CAMs and cohesion is usually moderate (but with quite high deviation) and certainly deserves further investigation. In terms of coupling (captured by MPC), "*Comments Area*" had the most striking correlation. This relation can be explained by supposing that the software engineer feels "*obliged*" to explain (through a comment) the need to call a method from another class. This mental procedure might explain the co-growth of comment areas and remote method calls, leading to a strong correlation with very low deviation. Finally, in term of complexity (captured with CC), the findings were very intuitive, in the sense that the "*Number of Conditionals*" is part of the calculation of cyclomatic complexity (simplistically counting `if`, `switch`, `for`, `while`, etc. statements in the code). Apart from this, CAMs also showcased a strong relation to complexity, since they are capturing indentations, which in well-styled code are used when conditional statements are implemented. We note that any attempt to use and interpret CAMs shall be made on code fragments that already conform to the basic code formatting principles (e.g., after running a code beautifier) or by employing an LLM (e.g., ChatGPT), feed it with a code fragment and request from the LLM to return a "*beautified*" version of the code.

6.2 Lessons Learnt: Implications to Researchers and Practitioners

Implications to Researchers: From a researcher's point of view, we can conclude that beauty metrics seem to be useful for quality assurance purposes. Therefore, we champion their further investigation in future studies. Our findings make clear that software engineers often look for specific 'anchors' in the code they read, consciously or subconsciously, which influences their assessment of overall aesthetic quality. Additionally, there is a need to study how the specific CAMs are changing when coding standards are applied, and when they are not. This study will be important for differentiating cases where lack of simplicity is due to lack of formatting and necessary lack of blank spaces, e.g., due to reduced complexity of the code chunk. To our perception, such a study would require the combination of code styling and code beauty metric, under a common model or tool. Moreover, it is important to study possible causality between beauty and quality, as well as tentative third causal factors, e.g., style or skills of the software engineer. Another interesting extension to this work would be to conduct a qualitative study in which the software engineers are asked "*why*" they perceive a code snippet as beautiful, and when not. Such work would provide explainability to our models and could lead to additional metrics, or updates of the existing ones. Moreover, we can replicate this study with additional programming paradigms and languages, to check the transferability of the findings outside object-orientation and Java code.

Finally, since the proposed beauty model is comprehensive as possible extensions, we foresee both the addition of extra beauty viewpoints, characteristics, or metrics. For instance, we believe that metrics that would be able to capture the compliance of variable and method names to internal styling conventions, or the proper instantiation of a design pattern would improve the completeness of the model—constituting them as an interesting future work direction.

Implications to Practitioners: Given the above findings, we can claim that practitioners' "*first look*" on a code fragment can act as a quite reliable approximation of the quality, if basic formatting standards are obeyed. In that sense, we believe that code beauty must be a concern of the software engineer, while writing the code, especially targeting on writing small, modular, and less complex methods. This rule of thumb follows some basic principles of object-orientation, such as the Open-Close Principle, the use of Polymorphism, the adoption of the Single Responsibility Principle, etc. Finally, based on the outcomes of RQ_2 we can propose some high-level guidelines, which software engineers can instantiate and consider along code beautification, to fit available tool-support and company internal regulations and working style:

- First, the developer must use some kind of beautifier support to fix styling issues that can be automated;
- Second, the developer must consider, check and validate the textual consistency of code, and the conformance to naming conventions of variables and methods;
- Third, the code must be Simple and Regular, with only the necessary intends and blank lines
- Finally, the developer must check the conformance to the following metric thresholds for approximately a screen of code (~30 lines):
    - the Commented Area must be more than 10% of the code (see Figure 16)
    - the class must not have more than 4 Conditional Statements (see Figure 14)
    - the class must not have more than 8 SonarQube issues (see Figure 15)

6.3 Threats to Validity

While our study endeavors to explore the correlation between code aesthetics and quality metrics comprehensively, it is essential to acknowledge potential threats to the validity of findings. First and foremost, the generalizability of our results may be limited by the specific set of people, projects and codebases chosen for analysis, as well as the use of only one programming language (i.e., Java) and one programming paradigm (i.e., object-orientation). The characteristics and coding practices of these projects might not be representative of the broader software development landscape posing threats to external validity. Further replication studies would be needed to validate the identified correlations between code beauty measures and quality attributes. The construct validity of the study is threatened by the choice of code quality metrics, as certain aspects of code quality may not be fully captured by the selected indicators. Additionally, the validity of the selected metrics is presented in Section 5.2, and those that have not proven to be significant can be replaced with others. Similarly, the concept of beauty is inherently subjective and as a result the employed measures of code beauty reflect only some aspects of aesthetics in text / code. It would be reasonable to assume that the subjective nature of aesthetic evaluations may introduce inter-rater variability (in the case of human assessments). The metrics employed for aesthetic evaluation might not encompass all dimensions of code beauty, and different stakeholders may have diverse opinions on what constitutes "beautiful" code. Furthermore, some aesthetic metrics may be counterintuitive for code development, as many were adapted from UI design metrics. Since source code naturally follows a left-skewed structure due to indentation and varying line lengths, metrics like Symmetry and Equilibrium, which aim for equal character distribution across all four quadrants, may not align with the inherent characteristics of writing code. While subjectivity has not been included as an independent parameter in our study, we attempted to mitigate this threat by substituting subjective evaluation with established code aesthetic metrics. Finally, we note that this study is not aimed at identifying causal relations but providing an initial exploration of correlations.

7. Conclusions

Confirming the correlation between code beauty and code quality represents a pivotal finding in software engineering research. Through a meticulous analysis of various aesthetic metrics and their correlation with code quality measures, our study validates the intuitive belief that beautifully crafted code aligns with higher overall software quality. Furthermore, the consensus of beauty in the eyes of software engineers and its correlation with code quality, reveals a nuanced understanding of what constitutes code beauty across diverse programming backgrounds and levels of expertise. By employing Spearman correlation, our research reveals the relationship between factors like code simplicity, regularity and metrics indicative of maintainability, performance, and reliability. The confirmation of this correlation underscores the importance of aesthetics in code development. It implies that codebases exhibiting elegance in design and structure are not only visually appealing but also tend to harbor qualities associated with robustness and maintainability.

Acknowledgements

This research is funded by the University of Macedonia Research Committee as part of the "Principal Research 2023" funding program.

References

Aldenhoven, C. M. and Engelschall, R. S., "The beauty of software architecture", *20th International Conference on Software Architecture (ICSA)*, L'Aquila, Italy, 2023, pp. 117-128, 2023.

Al-Saiyd, N. A., "Source code comprehension analysis in software maintenance", *2nd International Conference on Computer and Communication Systems (ICCCS)*, Krakow, Poland, 2017.

Arvanitou, E. M., Argyriadou, P., Koutsou, G., Ampatzoglou, A. and Chatzigeorgiou, A., "Quantifying TD Interest: Are we Getting Closer, or Not Even That?", *48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA' 22)*, IEEE Computer Society, Gran Canaria, Spain, August 2020.

Begel, A. and Nagappan, N. "Pair programming: what's in it for me?", *2nd ACM-IEEE International symposium on Empirical software engineering and measurement*, pp. 120-128, October 2008.

Buse R. P. L. and Weimer, W., "Learning a metric for code readability", *IEEE Transactions on Software Engineering*, 36 (4), pp. 546–558, 2010.

Campbell G. A, "Cognitive complexity: an overview and evaluation", *International Conference on Technical Debt (TechDebt '18)*, Gothenburg, Sweden, 27 - 28 May 2018.

Campbell, G.A. and Papapetrou, P.P., "SonarQube in action", *Manning Publications,* October 2013.

Cates, R., Yunik, N. and Feitelson, D. G., "Does Code Structure Affect Comprehension? On Using and Naming Intermediate Variables", *29th International Conference on Program Comprehension (ICPC)*, Madrid, Spain, 2021.

Chen, C., Alfayez, R., Srisopha, K., Boehm, B. and Shi, L., "Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It?," *39th International Conference on Software Engineering Companion (ICSE-C)*, Buenos Aires, Argentina, 2017, pp. 377-378, 2017.

Chidamber, S. R. and Kemerer, C. F., "A metrics suite for object-oriented design", *IEEE Transactions on software engineering*, 20(6), 1994.

Coleman, R., "Beauty and Maintainability of Code", *International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2018.

Connolly, A., Hellerstein, J.L., Alterman, N., Beck, D.A., Fatland, R., Lazowska, E., Mandava, V., and Stone, S., "Software Engineering Practices in Academia: Promoting the 3Rs—Readability, Resilience, and Reuse*", Issue 5.2*, Spring 2023.

Davila, N. and Nunes, I., "A systematic literature review and taxonomy of modern code review", *Journal of Systems and Software*, 177, 2021.

Dexter, S., Dolese, M., Seidel, A., and Kozbelt, A., "On the Embodied Aesthetics of Code", *Culture Machine*, 12 (1), 2011.

Duchess, T., "Molly Bawn", *Dodo Press*, 2008.

Eichelberger, H., "Nice class diagrams admit good design?", *ACM symposium on Software visualization (SoftVis '03)*, Association for Computing Machinery, New York, NY, USA, 2003.

Fowler, M., Beck, K. , Brant, J., Opdyke, W. and Roberts, D., "Refactoring: Improving the Design of Existing Code", *Addison-Wesley Professional*, 1st Edition, July 1999.

Gelernter, D., "Machine Beauty: Elegance And The Heart Of Technology", *Basic Books*, 1998.

Herraiz, I., Rodriguez, D., Robles, G. and Gonzalez-Barahona, J. M., "The evolution of the laws of software evolution: A discussion based on a systematic literature review", *ACM Computing Surveys (CSUR),* 46 (2), Article 28, pp. 1-28, November 2013.

Kendall, M. G., "Rank correlation methods", *Griffin*, Oxford, England, 1948.

Kursa, M. B. and Rudnicki, W. R., "Feature selection with the Boruta package", *Journal of statistical software*, 36, pp. 1-13, 2010.

Letouzey J.-L., "The SQALE method for evaluating technical debt", *3rd International Workshop on Managing Technical Debt (MTD '12)*, Zurich, Switzerland, 5 June 2012.

Li, W., and Henry, S., "Object-oriented metrics that predict maintainability", *Journal of systems and software*, 23(2), 111-122, 1993.

Loriot B., Madeiral F. and Monperrus, M., "Styler: learning formatting conventions to repair Checkstyle violations", *Empirical Software Engineering*, 27, 149, 2022.

Loukaki, A., "Living Ruins, Value Conflict", *Farnham: Ashgate*, 2008.

Lundberg, S. M., and Lee, S. I., "A unified approach to interpreting model predictions", *Advances in neural information processing systems*, 30, 2017.

Martin R.C., "Clean code", *Philadelphia, PA: Prentice Hall*, 2008.

McCabe, T., "A complexity measure", *IEEE Transactions on Software Engineering*, 2 (4), 308–320, 1976.

Ngo D. C. L., Teo L. S. and Byrne J. G., "Evaluating Interface Esthetics", *Knowledge and Information Systems*, 4 (1), pp. 46–79, 2002.

Nikolaidis, N., Mittas, N., Ampatzoglou, A., Arvanitou, E. M., and Chatzigeorgiou, A., "Assessing TD Macro-Management: A Nested Modeling Statistical Approach", *IEEE Transactions on Software Engineering*, 49(4), pp. 2996-3007, 2023.

Oliveira, D., Santos, R., Madeiral, F., Masuhara, H., and Castor, F., "A systematic literature review on the impact of formatting elements on code legibility", *Journal of Systems and Software*, 203, 2023.

Oram, A. and Wilson,vG., "Beautiful Code", *Newton, O' Reily*, 2007.

Riaz, M., Mendes, E., and Tempero, E., "A systematic review of software maintainability prediction and metrics", *3rd international symposium on empirical software engineering and measurement*, October 2009.

Santayana G., "The Sense of Beauty", *New York: Dover Publications*, 1955.

Scalabrino S., Linares-Vásquez M., Oliveto R., and Poshyvanyk D., “A Comprehensive Model for Code Readability”, *Journal of Software: Evolution and Process*, 30 (6), 2018.

Van Koten, C. and Gray, A. R., “An application of Bayesian net-work for predicting object-oriented software maintainability”, *Information and Software Technology*, 48(1), pp. 59-67, 2006.

Wertheimer M, Riezler K., “Gestalt Theory”, *Social Research*, 11 (1), pp 78–99, 1994.

De Winter, J. C., Gosling, S. D., and Potter, J., “Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data”. *Psychological methods*, 21(3), 273, 2016.

Zeki S., Romaya J. P., Benincasa D. M. T. and Atiyah M. F., “The experience of mathematical beauty and its neural correlates”, *Frontiers in Human Neuroscience*, 8 (68), 2014.

Zhou, Y. and Leung, H., “Predicting object-oriented software maintainability using multivariate adaptive regression splines”, *Journal of systems and software*, 80 (8), pp. 1349-1361, 2007.