Analyzing Programmer Psychological Traits Using Code and Sentiment Analysis

Jorge Martinez-Gil

Software Competence Center Hagenberg GmbH
Softwarepark 32a, 4232 Hagenberg, Austria
jorge.martinez-gil@scch.at

Abstract

Our research introduces a novel framework for analyzing programmer psychological traits by combining automated code analysis and sentiment assessment. Our contribution is in the form of a framework that can identify key behavioral traits such as attention to detail, collaboration orientation, and problem-solving tendencies. Unlike traditional survey-based methods, this approach offers objectivity and applicability across diverse programming contexts. Experimental results demonstrate its potential to contribute to software engineering, developer productivity research, and cognitive profiling of programmers.

Keywords: Software Engineering, Code Analysis, Sentiment Analysis, Developer Traits, Natural Language Processing

1. Introduction

Programmers' psychological traits significantly influence software quality, productivity, and team dynamics [2, 3]. However, traditional methods for analyzing these traits, such as surveys or observational studies, are often subjective and labor-intensive.

Our research examines how the psychological traits of programmers can be inferred by analyzing their source code and comments. The research uses techniques for code analysis combined with Natural Language Processing (NLP) to identify traits like attention to detail, collaboration tendencies, and problem-solving approaches. The process involves extracting and analyzing comments for sentiment, identifying specific keywords associated with particular behaviors, and measuring code features like complexity and adherence to standards [8]. Our strategy provides a structured approach to evaluate traits directly from programming activity, moving away from subjective survey-based assessments.

The research also explores practical uses in team organization and improving software development practices, including readability aspects [9]. Some novel techniques for code analysis

were employed to analyze source code, demonstrating the practicality of this approach. Future developments should be able to support different programming languages and improve trait identification accuracy through more advanced algorithms. These efforts aim to refine how behavioral characteristics are detected and interpreted from how programmers write and document their code.

Therefore, this study presents an automated framework to infer psychological traits directly from source code. We think that the major contributions that this novel approach provides are the following:

- Our framework can identify insights into key traits such as attention to detail, collaboration orientation, and problem-solving skills.
- Our framework can perform automated extraction of emotional tone from comments and developer notes.
- Our framework could lead to applications for team management, software quality assurance, and personalized learning tools.

The rest of this paper is organized to present its contributions in a clear and logical sequence. Section 2 provides an overview of existing studies, showing how this approach differs and adds value. Section 3 explains the technical contribution, ensuring readers can understand and reproduce the steps. Section 4 applies the method to real-world examples, demonstrating its practical applications. Section 5 examines the strengths and limitations of the work and proposes future improvements. Finally, we conclude with the major lessons that can be extracted from our study.

2. Related Work

Modern software development often relies on automated tools to generate code, reducing the amount produced entirely by human programmers [6]. This shift leaves many areas unexplored, including psychological signals inferred from a person's coding approach. Research has considered static and run-time checks to measure complexity, style, and error patterns. Recent work applies NLP and machine learning, examining commit messages, issue trackers, and code repositories. Building upon these foundations, the method described here uses sentiment analysis and keyword extraction to identify psychological signals.

This field has an existing base of literature. For example, previous efforts examined possible links between personality traits, coding habits, and performance [5]. Other research studies addressed conscientiousness in pair programming [13], academic settings [12], and the review of source code [2], along with similarity measures [10] and interpretability [7].

Moreover, various tools and systems are already available to study source code and examine programmer behavior. Static analysis software, such as SonarQube[1], measures metrics like code quality and complexity, offering a means to evaluate patterns in how developers approach coding tasks. Version control tools like GitMiner[2] allow researchers to explore commit histories and changes, shedding light on patterns in programming habits and team interactions. These platforms provide structured data, which can serve as the foundation for further behavioral examination and collaboration [1].

Additionally, plugins integrated into development environments, such as IntelliJ[3], gather information by monitoring activity patterns within these tools. It also appears feasible to consider semantic similarity [11] with previously developed solutions since mapping functions can be created to allow existing knowledge to be reused.

3. Methodology

Our approach explores various techniques to analyze programmer behavior by extracting patterns from source code and related activities. Static code analysis measures code complexity, style adherence, and maintainability, which provide clues about cognitive preferences and work habits. Sentiment analysis of comments helps identify emotional states and communication styles. Examining the frequency and type of edits in version control systems is intended to track problem-solving approaches and collaboration tendencies.

Other techniques include keyword analysis, which identifies words associated with specific traits, and bug report profiling to assess problem-handling strategies. Statistical and machine learning methods can further process this data to detect behavioral trends over time, although this is outside the scope of this paper. The idea is that, together, these methods can create a detailed view of how individual traits and team dynamics manifest through code and related interactions. Below, we offer a formal view of our framework.

3.1. Mathematical Representation of Framework

Let C represent the source code, K the set of keywords associated with specific traits, and M the comments. Let $\mathcal{S}(C)$ be the set of extracted comments, and $\mathcal{A}(M)$ the sentiment analysis function applied to the comments.

[1] `https://www.sonarsource.com/products/sonarqube/`
[2] `https://gitminer.com/`
[3] `https://www.jetbrains.com/idea/`

3.1.1. Static Code Analysis Metrics

We compute the following metrics:

$$\begin{aligned} \text{Complexity} : \mathcal{C}(C) &= f_1(C) \quad \text{(e.g., cyclomatic complexity)} \\ \text{Style Adherence} : \mathcal{T}(C) &= f_2(C) \quad \text{(e.g., style violations)} \\ \text{Maintainability} : \mathcal{M}(C) &= f_3(C) \quad \text{(e.g., maintainability index)} \end{aligned}$$

These metrics provide insights into programmer behavior:

$$\text{Behavioral Traits} \propto (\mathcal{C}(C), \mathcal{T}(C), \mathcal{M}(C)).$$

3.1.2. Sentiment Analysis

Sentiment analysis is applied to comments in the following way:

$$\text{Emotional Tone} = \mathcal{A}(M), \quad \mathcal{A} : M \to \mathcal{R}.$$

3.1.3. Edit Frequency and Collaboration Analysis

Define $E(t)$ as the frequency of edits over time t, and $P(e)$ as the type of edit $e \in E$. Problem-solving approaches are analyzed as follows:

$$\mathcal{P}(E) = g(E, P).$$

3.1.4. Keyword Analysis

Keyword identification is modeled as follows:

$$\mathcal{K}(C) = \{k \in K : k \text{ appears in } C\}.$$

3.1.5. Statistical and Machine Learning Methods

Behavioral trends over time t are modeled as:

$$\mathcal{B}(t) = h(\mathcal{C}, \mathcal{T}, \mathcal{M}, \mathcal{A}, \mathcal{K}).$$

3.1.6. Comprehensive Behavior Model

The overall behavioral model is constructed by integrating all components. Let $\mathcal{D}$ denote the dataset comprising all observations:

$$\mathcal{D} = \{\mathcal{C}(C), \mathcal{T}(C), \mathcal{M}(C), \mathcal{A}(M), \mathcal{K}(C), \mathcal{P}(E), \mathcal{U}(t)\}.$$

A unified model $\mathcal{H}$ that captures programmer behavior is then defined as:

$$\mathcal{H} = \phi(\mathcal{D}),$$

Where ϕ represents a machine learning or statistical function that extracts patterns and trends from $\mathcal{D}$ to predict cognitive preferences, emotional states, and collaboration tendencies.

4. Use Case

The methodology used data from real-world Python repositories to analyze programmer behavior. A specific code snippet was selected to demonstrate the approach: a short block of code extracted from a popular open-source library showcasing a typical error-handling pattern and the associated comments programmers left behind as they iteratively refined their implementation.

```python
# TODO: Refactor this function to improve readability
def calculate_sum(a, b):
    try:
        return a + b  # Adding two numbers
    except TypeError:
        # Fixme: Handle cases where input is not numeric
        print("Error: Inputs must be numbers")
        return None
```

Listing 1: Sample Code Snippet

1. **Code Structure and Style:** The complexity of the function was measured as $\mathcal{C}(C) = 2$, indicating simple logic. Style compliance was rated $\mathcal{T}(C) = 0.9$, reflecting adherence to coding conventions.
2. **Sentiment in Comments:** Comments were neutral in tone, with $\mathcal{A}(M) = 0.0$. Including *TODO* and *Fixme* hints at an active focus on improving and resolving issues.
3. **Keyword Detection:** Keywords such as *Refactor* and *Error* were identified, resulting in $\mathcal{K}(C) = \{\text{"Refactor"}, \text{"Error"}\}$. These terms point to a consideration for quality and error management.
4. **Patterns in Edits:** The frequency and nature of error handling and comments revealed a problem-solving tendency, calculated as $\mathcal{P}(E) = 0.75$. Collaboration patterns were moderate, represented by $\mathcal{U}(t) = 0.6$.

5. Discussion

Our research can also help identify practical applications for software development teams and project management. The idea behind analyzing programming behavior is to plan tasks that can be assigned more effectively, aligning with individual strengths to improve efficiency. Training programs can be tailored to address specific areas where developers may benefit from skill improvement. This method can also uncover workflow issues or patterns contributing to frequent errors, allowing teams to address problems before they escalate. These practices support better team organization, clearer role distribution, and streamlined processes in software projects.

The results demonstrate that programming habits and collaboration can be assessed using structured methods. Developers with complex code structures may focus on simplifying their solutions, and neutral comments indicate clearer communication opportunities. Patterns in using keywords suggest contributions to tasks like bug resolution and quality assurance.

Therefore, the analysis shows that programming behavior can guide improvements in individual and team processes. Recognizing coding patterns and interactions allows teams to adjust workflows to improve collaboration.

There are also challenges to this approach. Sparse or poorly written comments limit the accuracy of the analysis, and predefined keywords may not capture all relevant behaviors. In order to address these limitations, future work will include expanding the method to handle multiple programming languages and incorporating machine learning models for more precise detection of traits. Testing is an area that could also be further explored [4]. Moreover, it is an idea to proceed with validation using established psychological assessments to enhance the reliability of the findings and support its application in varied software development settings.

6. Conclusion

Our work has introduced an automated framework for examining programmer behavior by analyzing source code and comments. Our framework is intended to identify patterns that reveal behavioral tendencies such as attention to detail, communication habits, and approaches to problem-solving. This method shifts away from traditional manual evaluations, offering a more objective alternative.

The results also show us practical applications for improving team productivity, identifying training needs, and supporting managerial decisions in software development teams. Future efforts will address current limitations, such as expanding compatibility with multiple programming languages and improving the precision of trait detection through advanced modeling techniques. Collaboration with experts from behavioral sciences could further refine the method, enabling a deeper connection between code patterns and psychological traits. These advancements aim to provide organizations and researchers with actionable information to support the human aspect of programming.

Acknowledgments

The research reported in this paper has been funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation, and Technology (BMK), the Federal Ministry for Digital and Economic Affairs (BMDW), and the State of Upper Austria in the frame of SCCH, a center in the COMET - Competence Centers for Excellent Technologies Programme.

References

[1] Curtis, B., & Walz, D. (1990). The psychology of programming in the large: Team and organizational behaviour. In *Psychology of programming* (pp. 253–270). Elsevier. URL: https://www.sciencedirect.com/science/article/pii/B9780123507723500215. doi:https://doi.org/10.1016/B978-0-12-350772-3.50021-5.

[2] Da Cunha, A. D., & Greathead, D. (2007). Does personality matter? an analysis of code-review ability. *Communications of the ACM*, *50*, 109–112. URL: https://doi.org/10.1145/1230819.1241672. doi:10.1145/1230819.1241672.

[3] Hannay, J. E., Arisholm, E., Engvik, H., & Sjoberg, D. I. (2009). Effects of personality on pair programming. *IEEE Transactions on Software Engineering*, *36*, 61–80. URL: https://doi.org/10.1109/TSE.2009.41. doi:10.1109/TSE.2009.41.

[4] Kanij, T., Merkel, R., & Grundy, J. (2013). An empirical study of the effects of personality on software testing. In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)* (pp. 239–248). IEEE. URL: https://doi.org/10.1109/CSEET.2013.6595255. doi:10.1109/CSEET.2013.6595255.

[5] Karimi, Z., Baraani-Dastjerdi, A., Ghasem-Aghaee, N., & Wagner, S. (2016). Links between the personalities, styles and performance in computer programming. *Journal of Systems and Software*, *111*, 228–241. URL: https://doi.org/10.1016/j.jss.2015.09.011. doi:10.1016/J.JSS.2015.09.011.

[6] Martinez-Gil, J. (2023). A comparative study of ensemble techniques based on genetic programming: A case study in semantic similarity assessment. *Int. J. Softw. Eng. Knowl. Eng.*, *33*, 289–312. URL: https://doi.org/10.1142/S0218194022500772. doi:10.1142/S0218194022500772.

[7] Martinez-Gil, J. (2024). Advanced detection of source code clones via an ensemble of unsupervised similarity measures. *arXiv preprint arXiv:2405.02095*, . URL: https://doi.org/10.48550/arXiv.2405.02095. doi:10.48550/ARXIV.2405.02095.

[8] Martinez-Gil, J. (2024). Improving source code similarity detection through graphcodebert and integration of additional features. *arXiv preprint arXiv:2408.08903*, . URL: https://doi.org/10.48550/arXiv.2408.08903. doi:10.48550/ARXIV.2408.08903.

[9] Martinez-Gil, J. (2024). Optimizing readability using genetic algorithms. *Knowl. Based Syst.*, *284*, 111273. URL: https://doi.org/10.1016/j.knosys.2023.111273. doi:10.1016/J.KNOSYS.2023.111273.

[10] Martinez-Gil, J. (2024). Source code clone detection using unsupervised similarity measures. In P. Bludau, R. Ramler, D. Winkler, & J. Bergsmann (Eds.), *Software Quality as a Foundation for Security - 16th International Conference on Software Quality, SWQD 2024, Vienna, Austria, April 23-25, 2024, Proceedings* (pp. 21–37). Springer volume 505 of *Lecture Notes in Business Information Processing*. URL: https://doi.org/10.1007/978-3-031-56281-5_2. doi:10.1007/978-3-031-56281-5_2.

[11] Martinez-Gil, J., & Aldana-Montes, J. F. (2013). Semantic similarity measurement using historical google search patterns. *Inf. Syst. Frontiers*, *15*, 399–410. URL: https://doi.org/10.1007/s10796-012-9404-7. doi:10.1007/S10796-012-9404-7.

[12] Salleh, N., Mendes, E., & Grundy, J. (2014). Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments. *Empirical Software Engineering*, *19*, 714–752. URL: https://doi.org/10.1007/s10664-012-9238-4. doi:10.1007/S10664-012-9238-4.

[13] Salleh, N., Mendes, E., Grundy, J., & Burch, G. S. J. (2010). An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* (pp. 577–586). URL: https://doi.org/10.1145/1806799.1806883. doi:10.1145/1806799.1806883.