Can Personality Types Be Blamed for Code Smells?

Manoel Valerio da Silveira Neto[a], Andreia Malucelli[b] and Sheila Reinehr[c]
Graduate Program in Informatics (PPGIa), Pontifical Catholic University of Paraná, Block 8, Technological Park, 2nd floor, Imaculada Conceição Street, 1155 Prado Velho, Zip Code 80215-901, Curitiba - Paraná, Brazil

Keywords: Technical Debt, Code Smell, Personality Type, MBTI.

Abstract: The term code smell refers to sections of code that are not technically incorrect, do not prevent the software from functioning, but affect its quality. Code smells are considered a form of technical debt (TD). This study investigated the relationship between the personality types of software developers and the presence of code smells, which indicate potential problems in source code. Using the Myers-Briggs Type Indicator (MBTI) to classify personalities, the study examines whether specific profiles are more associated with creating or removing code smells. The goal is to assist software project managers in allocating tasks for refactoring and development. The research does not find a statistically significant correlation between the developer's personality and the creation of code smells. Still, it suggests that the Consul personality type (ESFJ) shows a greater tendency to resolve code smells. The study also highlights the importance of considering human factors such as personality types in software development to improve product quality.

1 INTRODUCTION

Software is a product resulting from human activities aimed at problem-solving, cognitive information processing, and social interaction (Sturdee et al., 2022). The software provides instructions and data for a computer to execute a specific task or perform a desired function. Software is created by developers who use programming languages and development tools to design and create programs that can be executed on different platforms.

Software, as a product distinctly originating from human activity, carries in its essence the marks of the capabilities, decisions, and peculiarities of the individuals who conceive and develop it. This intrinsic human nature of software means that its quality is directly affected by the characteristics and skills of the developers. Personality traits of individuals involved in the development process play a significant role in determining the final quality of the software (Capretz, 2003), (Capretz et al., 2010). These personality traits can influence programming practices and design decisions, leading to robust, well-structured software or software prone to failures and inefficiencies. Therefore, understanding these human nuances is essential for improving software development practices and product quality. Human nuance is understanding the complexities and varieties of human emotions, thoughts, and behaviors.

The personality and experience of the developer can influence the quality of the software (Ronald Laughery et al., 1985), (Capretz et al., 2010), (Fernández-Sanz et al., 2011), (Dutra et al., 2021), and one of the indicators of poor software quality is the presence of code smells.

The term code smell refers to sections of code that are not technically incorrect (Fowler et al., 2019), do not prevent the software from functioning, but affect its quality. Code smells are considered a form of technical debt (TD) of code smell. During the software development process, developers may inadvertently create code smells.

Code smells can be identified through static code analysis tools. Such tools can be combined with mining source code repositories, allowing developers to perform refactoring, as the static code analysis tool has identified and quantified the code smells.

[a] https://orcid.org/0000-0003-0470-5350
[b] https://orcid.org/0000-0002-0929-1874
[c] https://orcid.org/0000-0001-9430-7713

Silveira Neto, M., Malucelli, A. and Reinehr, S.
Can Personality Types Be Blamed for Code Smells?.
DOI: 10.5220/0012634700003690
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 2*, pages 196-205
ISBN: 978-989-758-692-7; ISSN: 2184-4992
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

Several studies have been conducted on the most common causes of technical debt (TD) due to code smells, among which are human factors. The research by Rios et al. (2018) shows that developers' lack of knowledge and experience can cause TD. The study by Kruchten et al. (2012) points out that the causes include pressure for deadlines (schedule), lack of care, insufficient education (a situation where the education provided is inadequate to satisfy the basic learning and development needs of an individual or group), deficient processes, non-systematic quality verification, or basic incompetence (referring to the lack of fundamental skills or knowledge needed to perform a specific task or function). Rios et al. (2019) identified that 21.9% of the problems are related to development, where the code smell is linked to the non-adoption of good coding practices (7.8%).

Code smells can indicate the need for additional training for the development team, communication problems within the group, or the need for better coding practices. Therefore, effective management of human factors can help minimize the occurrence of code smells, improve code quality, and, consequently, improve the quality of the software produced.

Research conducted by Capretz (2003), Beecham et al. (2008), Feldt et al. (2010), Gilal et al. (2017), and Sturdee et al. (2022) have investigated the personalities of software developers to understand how different personality characteristics and psychological traits can affect the development process and the quality of the code produced. However, there is no evidence of studies analyzing whether there is a relationship between technical debt from code smells and the developer's personality.

This work investigates whether specific personality profiles are associated with the creation or removal of code smells to assist software development project managers in allocating tasks related to refactoring or developing new features to specific developer profiles, thereby maximizing the quality of the code produced and minimizing the accumulation of technical debt from code smell.

This paper is organized as follows: Section 2 presents the theoretical background and Section 3 the related works; Section 4 describes the research method; Section 5 presents the results, which are discussed in Section 6; Section 7 presents the threats to validity; Section 8 concludes the article with perspectives for future work.

2 BACKGROUND

2.1 Code Smell

The concept of code smell was popularized by Kent Beck in the 1990s and explored in depth by Martin Fowler (Fowler et al., 2019). In Fowler et al. (2019), authors describe a series of patterns that can indicate problems in the code and offer strategies to deal with these problems. The term code smell describes code structures that, although technically correct, negatively affect the quality of the software. According to Kruchten et al. (2012), code smell is a technical debt (TD) encountered in software development. If addressed immediately, code smell can be easily avoided during implementation, but this depends on the developer's technical knowledge. According to Fowler et al. (2019), code smells are code structures that violate design and programming principles and indicate software problems that will require future maintenance activities through refactoring. Code smells are characteristics or signs that indicate potential problems or deficiencies in software's source code. They suggest something might be wrong with the code's design, structure, or implementation. Martin (2014) and Fowler et al. (2019) present examples of code smells, including code duplication, long methods, and God classes. These code smells can be detected through static code analysis tools.

According to McConnell (2013), technical debt (TD) can be classified as intentional or unintentional. Intentional TD is consciously assumed as a strategic tool where the benefits outweigh the consequences of the incurred TD at the time. Unintentional TD, on the other hand, is when technical debt is incurred accidentally and unconsciously.

Although a low quantity of code smells may not be considered a problem when they accumulate, they can lead to difficulties in the maintenance and evolution of the software (Fowler et al., 2019). They can increase the complexity of the source code, make changes difficult or risky to perform, and hinder the understanding of the source code. Thus, it is essential to be aware of code smells and address them appropriately to ensure that the source code remains easy to maintain and evolves over time. One of the ways to ensure the maintainability of source code is through refactoring (Fowler et al., 2019).

Refactoring is a disciplined way of cleaning up source code to minimize the chances of introducing bugs. According to Fowler et al. (2019), refactoring is the process of changing a software system so that it does not alter the external behavior of the source code but improves its internal structure. Refactoring the source code is paying technical debt or removing code smells. It requires developer time to correct, leading to rework effort. According to Bourque et al. (2014), 40 to 60% of maintenance tasks are dedicated to understanding the source code in which maintenance is performed. This understanding is directly associated with code design problems and code smells.

Some static analysis tools quantify code smells intending to demonstrate the total effort required to solve the problems. There are various static analysis tools, with SonarQube being the most well-known in the software industry. The study by Guaman et al. (2017) identifies SonarQube as a comprehensive tool for static code analysis, mainly because it adheres to the SQALE method. The SQALE method presented by Letouzey et al. (2012) uses the ISO/IEC 25010:2011 standard (ISO, 2011) as a reference. Each capability pointed out in the SQALE method is associated with a characteristic or sub-characteristics of the standard. A code smell is part of the maintainability index of the code SQI (Software Quality Index), addressed in SQALE. This index is available in SonarQube through the sum of the code smell's remediation effort (time).

In Silveira Neto et al. (2021), the authors apply data mining techniques to a source code repository to identify code smells, defining a process called TDMining. However, they do not recognize a data dimension for the developer's personality. The execution of the TDMining process enables the acquisition of code smell data by each developer from the utilized source code repositories. The process provides SQL and Python scripts for analyzing association rules, moving averages, and time series of code smell data using data obtained from SonarQube (Guaman et al., 2017) and PyDriller (Spadini et al., 2018). The TDMining process, in addition to observing code smells by project, also provides a dimensional data model with which it is possible to analyze data by the developer.

2.2 Psychological Personality Types

Gulati et al. (2015), the authors investigate the use of the Myers-Briggs Type Indicator (MBTI) (Myers et al., 1988) and Five-Factor Model (FFM) (Tupes et al., 1992) personality models in software engineering. The MBTI categorizes personalities based on four dimensions, aiding in understanding approaches to problem-solving and communication. At the same time, the FFM focuses on five main traits to understand their impact on developers' performance and satisfaction. The research by Gulati et al. (2015) highlights the importance of these models in identifying personality traits favorable to specific roles in industry, contributing to better performance and effectiveness in teamwork. They emphasize the need for more studies in this area to enhance effectiveness in software engineering continually.

The MBTI model will be used in this research once it is the most utilized model in software engineering research for personality analysis, as presented by Cruz et al. (2015). Their systematic mapping aimed to explore the influence and role of personality in software engineering. The systematic mapping does not address why the MBTI is the most used test in the selected articles.

The study conducted by Delgado et al. (2022) explores the impact of personality on software development, highlighting that personality has a positive effect on various tasks and processes in this area. However, the lack of consistency in the results of these studies raises questions about their validity, emphasizing the importance of more rigorous studies with the aid of human behavior experts. The research identifies the most used and reliable psychological models and instruments for assessing the personalities of software developers. MBTI is recognized as the most popular. However, in recent years, the FFM and its psychometric tools, such as the Big Five Inventory (BFI) (John, 1991) and NEO Five-Factor Inventory (NEO-FFI) (Costa et al., 1992), have gained more relevance in the field. The study points to the need for a more in-depth and comparative investigation between these different instruments and models. This is essential to reduce contradictions in future research, allowing for a more precise evaluation of the personality of engineers and software developers from a psychological perspective.

MBTI is a tool presented by Myers et al. (1988) as a self-report instrument designed to identify, through a questionnaire, how a person perceives the world and makes decisions. The questionnaire was based on the typological theory proposed by Carl Gustav Jung. MBTI divides psychological types into 16 distinct types, each represented by a four-letter code that describes a person's main preferences in four dimensions, resulting from the combination of four pairs of characteristics or psychological traits:

• Extraversion (E) vs. Introversion (I), where Extraversion (E) refers to people who prefer to energize by interacting with others and the external world, and Introversion (I) relates to people who like to energize by spending time alone or with a small group of close people.

• Sensing (S) vs. Intuition (N), where Sensing (S) refers to people who rely on concrete and tangible information perceived through the five senses, and Intuition (N) refers to people who rely on instincts and abstract or theoretical knowledge, looking at the whole picture.

• Thinking (T) vs. Feeling (F), where Thinking (T) refers to people who make decisions based on logic and objective analysis, and Feeling (F) refers to people who make decisions based on personal values and how decisions will affect others.

• Judging (J) vs. Perceiving (P), where Judging (J) refers to people who prefer a structured and decided lifestyle, and Perceiving (P) refers to people who prefer a flexible and adaptable lifestyle.

The MBTI presents the dichotomy between the Introverted and Extroverted psychological traits as a spectrum. At the extreme of Introversion (I), someone might be very reserved and private, while at the extreme of Extroversion (E), someone might be exceptionally open and gregarious. Most people are not at the extremes but somewhere in the middle of the spectrum, showing traits of both preferences in different situations. Concerning the other psychological characteristics of the MBTI, such as Sensing-Intuition, Thinking-Feeling, and Judging-Perceiving, they are also seen as spectrums. Each person may lean towards one preference over the other but often exhibits characteristics of both sides, depending on the context. This complexity reflects the dynamic nature of personality.

These 16 psychological traits are often grouped into four categories of personality known as the four temperaments of the MBTI, based on shared characteristics:

Sentinels (SJ) value security, stability, tradition, and being practical, organized, and responsible. ISTJ - Logistician: practical, reliable, and systematic. ISFJ - Defender: dedicated, warm, and careful. ESTJ - Executive: organized, assertive, and efficient. ESFJ - Consul: sociable, caring, and popular.

Diplomats (NF) are motivated by values and vision, seeking meaning and possibilities. They tend to be empathetic, compassionate, and creative. INFJ - Advocate: idealistic, mystical, and people-oriented. INFP - Mediator: idealistic, curious, and ethical. ENFJ - Protagonist: charismatic, inspiring, and selfless. ENFP - Activist: enthusiastic, creative, and sociable.

Analysts (NT) are oriented towards knowledge and competence. They tend to be innovative, strategic, and logical. INTJ - Architect: strategic, logical, and innovative. INTP - Logician: innovative, curious, and theoretical. ENTJ - Commander: charismatic, leader, and assertive. ENTP - Debater: inventive, intelligent, and insightful.

Explorers (SP): are realistic and action-oriented. They tend to be adaptable, spontaneous, and focused on the present moment. ISTP - Virtuoso: experimental, bold, and practical. ISFP - Adventurer: artistic, curious, and exploratory. ESTP - Entrepreneur: energetic, perceptive, and direct. ESFP - Entertainer: spontaneous, energetic, and enthusiastic.

The study by Capretz et al. (2010) concludes that there is a correlation between personality types and software development and that understanding the personality types of software developers can be helpful to team formation, task allocation, and project management. Moreover, the paper indicates that specific software development tasks may be more suitable for certain personality types. The importance of considering personality types in software development is also emphasized in the article as something that can contribute to more effective and efficient development processes. The report further highlights that most programmers possess characteristics of Introversion (I), Sensing (S), and Thinking (T) and concludes that analyzing these psychological characteristics when assigning people to stages of the software life cycle increases the chances of a successful project outcome. Additionally, the paper points out that the Sensing (S) and Perceiving (P) personality types are more suited for detail-oriented tasks and dealing with the constant changes inherent in software maintenance.

The study presented by Barroso et al. (2016) concludes that evidence suggests that the MBTI can be applied to software developers to understand how human personality influences the work of professionals. The research found that developers with MBTI type INTJ showed lower levels of Depth of Inheritance Tree (DIT) and slightly smaller methods (LOC). However, the paper also suggests that more research is needed to understand the relationship between personality and object-oriented software metrics.

3 RELATED WORK

To understand related works, an exploratory, semi-structured search was conducted on the scientific databases Scopus, IEEE, ACM, and SpringerLink with the search string: ("SOFTWARE ENGINEERING") AND ("PERSONALITY" OR "MBTI" OR "Briggs MYERS" or "Briggs-MYERS") AND ("CODESMELL" OR "TECHNICAL DEBT").

The search returned two articles: (Graf-Vlachy et al., 2023) and (Huang et al., 2021).

Graf-Vlachy et al. (2023) analyzed how the developer's personality affects the accumulation of technical debt (TD), suggesting a correlation between personality traits and decisions leading to TD. The article does not mention the MBTI concerning the developer's personality and technical debt. Still, it examines the relationship between technical debt and various characteristics of the developers' personalities, using the Five Factor Model (FFM), regulatory focus, and narcissism. These personality characteristics were evaluated in a software engineering context to understand their influence on introducing and removing technical debt.

The article by Huang et al. (2021) addresses developers' feelings about community smell, which is unrelated to the present research.

The research presented in this article aligns with the study of Capretz et al. (2010), as it aims to investigate the psychological traits of the developer concerning the creation and removal of code smells. The investigation becomes necessary to collaborate in identifying the profiles of creators or non-creators of code smell, thus adapting developers to specific activities of refactoring or new developments.

4 RESEARCH METHOD

The research was operationalized following the steps presented in Figure 1, described below.

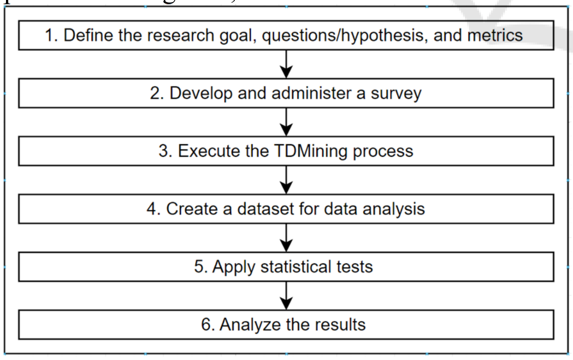
1. Define the research goal, questions/hypothesis, and metrics
2. Develop and administer a survey
3. Execute the TDMining process
4. Create a dataset for data analysis
5. Apply statistical tests
6. Analyze the results

Figure 1: Research steps.

(1) Define the research goal, questions/hypothesis, and metrics: **the research goal** was defined according to the Goal-Question-Metric (GQM) approach (Basili et al., 1988) as follows:
Goal: Identify if there is a statistical correlation between the quality of source code measured by code smells and the personality types of developers, based on the MBTI, to investigate whether specific types or categories of personality tend to create, or not, code smells, concerning the quality of source code, from the perspective of software developers, in the context of personality types in software development.
Question: Is there a statistical correlation between the quality of source code (code smell) and the personality types of developers (MBTI)?
The null hypothesis H0 is that there is no correlation between the quality of the source code and the developer's personality. The alternative hypothesis HA is that there is a correlation between the quality of source code and the developer's personality.
Metric: This involves statistically evaluating the correlation between code smell and the developer's personality. In the case of the relationship between the quality of source code and the developer's personality, the null hypothesis suggests that there is no correlation between the variables of interest (the quality of source code and developer's personality) and that any observed difference in the data is due to chance or other factors not related to the variables of interest.
(2) Develop and administer a survey: To conduct the research, it was necessary to collect the personality type of software developers. Therefore, a survey was designed, including questions aimed at discovering the MBTI profile of the professional. Additionally, information was collected to characterize the developers through questions, as shown in Table 1.

Table 1: Survey Questionnaire.

Question	Possible Answers
What type of development do you work in? (Developer_Specialty)?	BackEnd, FrontEnd, Mobile, FullStack, Other.
What is your career level? (Developer_Level)?	Junior, middle, senior, specialist, Other.
How many months of experience do you have with software development? (Developer_Month_Experience)?	Months
How many months have you been working at the company? (Developer_Month_In_Company)?	Months
What is your profile on the website https://www.16personalities.com/ (MBTI)? On the website, developers answer questions to identify their MBTI profile. Upon completing the questionnaire, developers attach an image showing their MBTI profile to the form.	Personality

Data regarding the developers' profiles were extracted from the questionnaire. The questionnaire was electronically sent to 50 software developers from a financial institution. Voluntarily, 40 developers responded to the questionnaire.

All developers participated voluntarily and signed an Informed Consent Form (ICF).

(3) Execute the TDMining process: This step involves mining the source code repository following the TDMining process Silveira Neto et al. (2021). The source code was mined from four years across 163 repositories. The mining process was not aimed at analyzing the types of code smells created by developers but at quantifying the occurrence of code smells created and removed. There were 216 developers in the entire source code versioning history. At the time of the research, the company had 50 developers. We analyzed the code smells of the company's active developers when the study was conducted.

(4) Create a dataset for data analysis: In this stage, the questionnaire data containing developer personality traits and the data from the mining of code smells created and removed by the developer were merged for potential analysis. The merged data were organized into two types of variables: numerical and categorical.

The numerical variables are:

- Developer_Month_Experience
- Developer_Month_In_Company
- Code_Smell_Created
- Code_Smell_Removed
- Code_Smell_Balance
- Developer Age

The dataset is available at https://doi.org/10.5281/zenodo.10072712. The variables that make up the dataset are:

- Type_MBTI: Developer's MBTI personality type.
- Developer_Gender: Developer's gender.
- Developer_Specialty: Developer's specialty.
- Developer_Level: Developer's experience level in the company.
- Developer_Month_Experience: Number of months of developer's experience.
- Developer_Month_In_Company: Number of months the developer has been with the company.
- Developer Age: Age of developer.
- MBTI: Detailed information about Type_MBTI.
- E_I: Extroversion/Introversion dimension of MBTI.
- S_I: Sensing/Intuition dimension of MBTI.
- T_F: Thinking/Feeling dimension of MBTI.
- J_P: Judging/Perceiving dimension of MBTI.
- Code_Smell_Created: Number of code smells created by the developer.
- Code_Smell_Removed: Number of code smells removed by the developer.
- Code_Smell_Balance: Code smells balance, i.e., the difference between code smells created and removed.
- Code_Smell_Action: Classification of the developer as a creator, neutral, or remover of code smells.

While creating the dataset, data anonymization was carried out, which consisted of removing names and data that could identify the developers.

(5) Apply statistical tests: Descriptive statistics were used to characterize the developers, and the chi-squared test was used to analyze the correlation between the variables. The chi-squared test is a standard statistical tool for analyzing contingency tables. It is often used to assess whether observed patterns or outcomes are due to chance or a real relationship, which is necessary when seeking associations in categorical data. In addition to the chi-squared test, Cramer's V, a measure of association between two nominal (categorical) variables based on the chi-squared test, was used. It helps determine the strength and significance of the association between two categorical variables. Regarding the interpretation of Cramer's V values, a measure close to 0 suggests either no or very little association between the variables, and a value close to 1 indicates a strong association between the variables.

The correlation between Code_Smell_Action and Type_MBTI, Developer_Gender, Developer_Specialty, Developer_Level, MBTI, E_I, S_I, T_F, and J_P was analyzed.

(6) Analyze the results: Examine the statistical test results and either confirm or reject the hypotheses based on these results.

5 RESULTS

The results are organized based on participant characteristics, the statistical correlation between code smell action (Code_Smell_Action) and personality, and developers' personality concerning code smell action.

5.1 Characterization of the Developers

Table 2 presents the characterization results of the software developers who participated in the research. The first column of Table 2 displays the name of the analyzed variable, as obtained in the dataset. In contrast, the second column shows the mean, followed by the minimum and maximum values for the variable.

Table 2: Characterization of the developers.

Feature	Mean	Min	Max	STD
Developer Month Experience	98.5	5	252	69.3
Developer Month In Company	23.5	2	140	22.5
Code Smell Created	188.2	0	4767	754.13
Code Smell Removed	27.8	0	325	57.8
Code_Smell Balance	160.3	185	4609	732
Developer Age	30.6	21	45	6.1

It can be observed that, on average, developers have 98.5 months of experience, and the time these developers have been with the researched company is 23.5 months. Developers, on average, created 188.22 code smells and removed an average of 27.87 code smells. The average age of developers is 30 years.

Out of the 40 developer records available in the dataset, 14 psychological traits out of 16 possible were identified. The two unidentifiable traits were the Activist (ENFP) and Entrepreneur (ESTP), and the most frequent psychological trait is ESFJ (Consul), accounting for 8 cases, which represents 20% of the developers.

Regarding the gender of the developers (Developer_Gender), out of the 40 developers, 33 (82.5%) are male. Regarding the specialty of the developers (Developer_Specialty), out of the 40 developers, 29 work with Backend, 3 with FrontEnd, 4 with Mobile development, and 4 with FullStack development. Analyzing the developer's level (Developer_Level), out of the 40 developers, 3 are specialists, 3 are interns, 2 are juniors, 13 are intermediate (middle), and 19 are senior developers.

Concerning standard deviation (STD), data reveals a comprehensive picture of the dynamics of developers and code quality in a company. On average, developers accumulate around 98.5 months of experience, with a moderate dispersion around this average, while their average stay in the company is approximately 23.5 months, with a relatively low variability. About code smells, the average creation is significantly high, with considerable dispersion, suggesting a wide range in the amount created per developer. In contrast, the average number of code smells removed is moderate, with a more controlled variation between developers. Finally, the average balance between "code smells" created and removed is positive, indicating a general trend towards accumulation, with considerable variability in the difference between developers.

5.2 Examining the Impact of Personality on Code Smell: A Statistical Analysis

Table 3 presents the result of all categorical variables concerning the variable Code_Smell_Action. For all categorical variables (column variable), the null hypothesis (column Rejected_H0) of no association was not rejected, which means that there is not enough evidence to assert a significant relationship between Code_Smell_Action and the other categorical variables. In other words, creating or removing code smells is unrelated to the developer's personality, specialty, or seniority.

The data were also evaluated for correlation using Cramer's V test (Table 3, Cramer's V column), which indicated that rejecting the null hypothesis is impossible. The Cramer's V result, being less than 0.3 for all variables, suggests a weak association between all the variables and Code_Smell_Action. According to the chi-squared test, this means there is a weak

Table 3: Statistical correlation between Code_Smell _Action and MBTI and other variables.

Variable	Cramer's V	p_value	Rejected_H0	Chi_Square
Type_MBTI	0.000	0.954	False	15.152
Developer_Gender	0.000	0.425	False	1.706
Developer_Specialty	0.247	0.094	False	10.805
Developer_Level	0.233	0.136	False	12.351
MBTI	0.000	0.954	False	15.152
E_I	0.283	0.074	False	5.188
S_I	0.000	0.438	False	1.646
T_F	0.000	0.570	False	1.123
J_P	0.273	0.083	False	4.976

statistical relationship between the analyzed variables.

The results shown in Table 3 were obtained using the IBM SPSS tool, where, in addition to the Cramer's V, p_value, Rejected_H0, and Chi_Square variables, it was possible to get contingency tables. Given this article's limited number of pages, the result can be replicated using the data available on Zenodo.

5.3 Personality Types and Their Role in Code Smell Creation and Removal: Insights from MBTI Analysis

Table 4 presents the quantitative result of personality types concerning code smell creation (creator, neutral, remover).

It is noticeable that the MBTI type "Consul" stands out in code smell removal, with the highest count of three (3) developers as code smell removers. Additionally, two developers remained with Code_Smell_Action equal to "Neutral." Being classified as "Neutral" for code smell creation does not imply that they have not contributed code to the source code repository. It means the same amount of code smell created equals the amount removed.

The TDMining process by Silveira Neto et al. (2021) requires source code analysis for each commit to the source code repository.

Therefore, submitting code to the source code repository does not necessarily imply the presence or removal of code smell.

Table 4: Type_MBTI x Code_Smell_Action.

Type_MBTI	Code_Smell_Action		
	Creator	Neutral	Remover
Advocate (INFJ)	4		
Entertainer (ESFP)	1		
Architect (INTJ)	2		1
Adventurer (ISFP)	1		
Commander (ENTJ)	1		
Consul (ESFJ)	4	1	3
Defender (ISFJ)	4		2
Executive (ESTJ)			1
Debater (ENTP)	1		
Logician (INTP)	3		
Logistician (ISTJ)	2		1
Mediator (INFP)	2		
Protagonist (ENFJ)	2	1	2
Virtuoso (ISTP)	1		

6 DISCUSSIONS

The results generally indicate no statistical correlation between a developer's personality and whether they are a code smell creator.

However, the Consul personality type (ESFJ) appeared more attentive to resolving code smells.

While this article focuses on the absence of a direct correlation between code smells and personality, related studies suggest that the influence of personality extends far beyond, affecting aspects such as technical debt, team effectiveness, and development process efficiency.

This indicates that, despite the specific findings of the article, the relationship between personality and software development is diverse and deserves further investigation, especially in the context of human factors in software engineering.

Graf-Vlachy et al. (2023) investigated the relationship between developers' personality traits (focusing on narcissism, using the FFM) and their influence on technical debt. The study examined 2,145 source code commits from 19 developers and aimed to answer how a developer's personality relates to introducing and removing technical debt.

The authors concluded that personality influences the introduction of technical debt, but they noted that the sample size was insufficient.

For the present study, there were no issues regarding the validity of the research since there were 23,628 commits, including 31,754 code smells involving 216 developers. Of these 216 developers, 50 were active in the company at the time of the research, and 40 responded to the personality survey.

Among the 40 developers who responded to the survey, 38 created or removed code smells (the other two developers neither created nor removed code smells, indicating a neutral role). These 38 developers accounted for 7,529 code smells created and 1,115 code smells removed in 5,046 commits.

Regarding Silveira Neto et al. (2021) work, the authors do not address the developer's personality as an additional dimension in the TDMining data model. Their work is limited to the relationship between code smells and personality, unlike the research presented in this article, which adds a data dimension for developer personality to the TDMining data model to correlate personality information.

The study by Capretz et al. (2010) demonstrates a correlation between personality types and software development, emphasizing the importance of personality in team formation and project management.

Capretz expands this discussion, suggesting that personalities can significantly influence the efficiency and effectiveness of software development. However, the provided sources did not mention specific details about the number of developers involved in the research or the study's evidence and weaknesses.

The research conducted by Dutra et al. (2021) does not find a direct correlation between code smells and personality profiles. Still, it suggests that the influence of personality in software development should be explored, particularly concerning human factors in software engineering.

Like Dutra et al. (2021), the present article suggests that software organizations, researchers, and professionals can benefit from understanding human factors to improve software quality.

7 THREATS TO VALIDITY

Regarding internal validity, given the size of the target population, with 40 developers representing 80% of the developers in the researched company, it is considered that the research is valid, primarily due to the difficulty in finding companies with a significant number of developers willing to respond to the questionnaire and possessing source code repositories for code smell mining and static code analysis tools.

Regarding external validity, it cannot be asserted that the results will be the same in other companies. About the conclusion validity, for the analyzed context and the population of developers, it is possible to state that the obtained results fulfill the objective within the scope of addressing the research question and hypotheses presented.

Concerning the construct validity, the result cannot be generalized; however, the applied approach can be replicated in other contexts.

Finally, concerning reliability, the research is reliable as it aligns with the body of work on personality in software engineering, using the MBTI, which Capretz has studied for decades. All the necessary steps for executing the process, along with anonymized data, have been provided, and the TDMining process is also available on GitHub.

8 CONCLUSIONS

Since there is not enough evidence to reject the null hypothesis, it is concluded that there is no significant association between MBTI profile variables and the action of creating code smells, meaning that personality types cannot be held accountable for code smells.

However, when examining the dataset and the sample size and conducting quantitative comparisons between profile groups, there are indications that the Consul profile type (ESFJ) shows concern regarding code smells. This research can help identify suitable personality profiles to assist project managers in assigning tasks related to refactoring or new functionalities to specific profiles.

Despite statistical tests showing no correlation, the profile identification procedure (survey) and the TDMining process can be used by other companies, and the provided data can be utilized for comparisons.

As prospects for future work, we are replicating the research with a more significant number of developers to enable potential results generalization.

ACKNOWLEDGEMENTS

The present work was conducted with the support of the Coordination for the Improvement of Higher Education Personnel (CAPES) - Funding Code 001, Brazil.

REFERENCES

Barroso, A. S., Madureira, J. S., Melo, F. S., Souza, T. D. S., Soares, M. S., & do Nascimento, R. P. C. (2016). An evaluation of influence of human personality in software development: An experience report. *2016 8th Euro American Conference on Telematics and Information Systems (EATIS)*, 1–6. https://doi.org/10.1109/EATIS.2016.7520108

Basili, V. R., & Rombach, H. D. (1988). The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering*, *14*(6), 758–773. https://doi.org/10.1109/ 32.6156

Beecham, S., Baddoo, N., Hall, T., Robinson, H., & Sharp, H. (2008). Motivation in Software Engineering: A systematic literature review. In *Information and Software Technology* (Vol. 50, Issues 9–10, pp. 860–878). https://doi.org/10.1016/j.infsof.2007.09.004

Bourque, P., & Fairley, R. E. (2014). SWEBOK v.3 - Guide to the Software Engineering - Body of Knowledge. In *IEEE Computer Society*. https://doi.org/10.123 4/12345678

Capretz, L. F. (2003). Personality types in software engineering. In *Int. J. Human-Computer Studies* (Vol. 58, pp. 207–214).

Capretz, L., Fernando, A., & Dr, F. (2010). *Making Sense of Software Development and Personality Types*.

https://ir.lib.uwo.ca/electricalpubhttps://ir.lib.uwo.ca/electricalpub/2

Costa, P., & Mccrae, R. (1992). Neo PI-R professional manual. *Psychological Assessment Resources*, *396*.

Cruz, S., Da Silva, F. Q. B., & Capretz, L. F. (2015). Forty years of research on personality in software engineering: A mapping study. *Computers in Human Behavior*, *46*, 94–113. https://doi.org/10.1016/j.chb.2014.12.008

Cunningham, W. (1992). The WyCash portfolio management system. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, *Part F1296*(October), 29–30. https://doi.org/10.1145/157710.157715

Delgado Jojoa Juan David and Revelo Sánchez, O. and C. S. V. (2022). Psychological Models and Instruments Employed to Identify Personality Traits of Software Developers: A Systematic Mapping Study. In P. H. and C.-M. O. Agredo-Delgado Vanessa and Ruiz (Ed.), *Human-Computer Interaction* (pp. 146–161). Springer International Publishing.

Dutra, E., Diirr, B., & Santos, G. (2021). Human Factors and Their Influence on Software Development Teams - A Tertiary Study. *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, 442–451. https://doi.org/10.1145/3474624.3474625

Feldt, R., Angelis, L., Torkar, R., & Samuelsson, M. (2010). Links between the personalities, views, and attitudes of software engineers. *Information and Software Technology*, *52*(6), 611–624. https://doi.org/10.1016/j.infsof.2010.01.001

Fernández-Sanz, L., & Misra, S. (2011). Influence of Human Factors in Software Quality and Productivity. *Proceedings of the 2011 International Conference on Computational Science and Its Applications - Volume Part V*, 257–269.

Fowler, M. (2019). *Refactoring Improving the Design of Existing Code Second Edition*.

Gilal, A. R., Jaafar, J., Abro, A., Umrani, W. A., Basri, S., & Omar, M. (2017). Making programmer effective for software development teams: An extended study. *Journal of Information Science and Engineering*, *33*(6), 1447–1463. https://doi.org/10.6688/JISE.2017.33.6.4

Graf-Vlachy, L., & Wagner, S. (2023). The Type to Take Out a Loan? A Study of Developer Personality and Technical Debt. *2023 ACM/IEEE International Conference on Technical Debt (TechDebt)*, 27–36. https://doi.org/10.1109/TechDebt59074.2023.00010

Guaman, D., Sarmiento, P. A. Q., Barba-Guamán, L., Cabrera, P., & Enciso, L. (2017). SonarQube as a tool to identify software metrics and technical debt in the source code through static analysis. *2017 7th International Workshop on Computer Science and Engineering, WCSE 2017*, *July*, 171–175. https://doi.org/10.18178/wcse.2017.06.030

Gulati, J., Bhardwaj, P., & Suri, B. (2015). Comparative Study of Personality Models in Software Engineering. *Proceedings of the Third International Symposium on Women in Computing and Informatics*, 209–216. https://doi.org/10.1145/2791405.2791445

Huang, Z., Shao, Z., Fan, G., Gao, J., Zhou, Z., Yang, K., & Yang, X. (2021). Predicting Community Smells' Occurrence on Individual Developers by Sentiments. *IEEE International Conference on Program Comprehension*, *2021-May*, 230 – 241.

Iso, I. O. F. S. (2011). Iso/Iec 25010:2011. *Software Process: Improvement and Practice*.

John, O. P. (1991). The Big Five inventory—versions 4a and 54. *(No Title)*.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, *29*(6), 18–21. https://doi.org/10.1109/MS.2012.167

Letouzey, J. L. J. L. J.-L., & Ilkiewicz, M. (2012). Managing technical debt with the SQALE method. *IEEE Software*, *29*(6), 44–51. https://doi.org/10.1109/MS.2012.129

Martin, R. C. (2014). Clean Code - A Handbook of Agile Software Craftmanship. In *Igarss 2014*. https://doi.org/10.1007/s13398-014-0173-7.2

McConnell, S. (2013). Managing Technical Debt (White Paper). In *Workshop on Managing Technical Debt (part of ICSE 2013)*.

Myers, I. B., & McCaulley, M. H. (1988). *Myers-Briggs Type Indicator: MBTI.* Consulting Psychologists Press.

Rios, N., Mendonça, M., Seaman, C., & Spínola, R. O. (2019). Causes and effects of the presence of technical debt in agile software projects. *25th Americas Conference on Information Systems, AMCIS 2019*.

Rios, N., Spínola, R. O., Mendonça, M., & Seaman, C. (2018). The most common causes and effects of technical debt: First results from a global family of industrial surveys. *International Symposium on Empirical Software Engineering and Measurement*. https://doi.org/10.1145/3239235.3268917

Ronald Laughery, K., & Laughery, K. R. (n.d.). *Human Factors in Software Engineering: A Review of the Literature*.

Silveira Neto, M. V., Reinehr, S., & Malucelli, A. (2021). TDMINING Process. In *GitHub repository*. https://github.com/manoelvsneto/tdmining.

Spadini, D., Aniche, M., & Bacchelli, A. (2018). PyDriller: Python framework for mining software repositories. *ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. https://doi.org/10.1145/3236024.3264598

Sturdee, M., Ivory, M., Ellis, D., Stacey, P., & Ralph, P. (2022). Personality Traits in Game Development. *ACM International Conference Proceeding Series*, 221–230. https://doi.org/10.1145/3530019.3530042

Tupes, E. C., & Christal, R. E. (1992). Recurrent personality factors based on trait ratings. *Journal of Personality*, *60 2*, 225–251.