# A preliminary investigation of developer profiles based on their activities and code quality: Who does what?

Cristina Aguilera González[†]
*Universitat Politècnica de Catalunya*
Barcelona, Spain
cristina.aguilera.gonzalez@estudiantat.upc.edu

Laia Albors Zumel[†]
*Universitat Politècnica de Catalunya*
Barcelona, Spain
laia.albors@estudiantat.upc.edu

Jesús Antoñanzas Acero[†]
*Universitat Politècnica de Catalunya*
Barcelona, Spain
jesus.maria.antonanzas@estudiantat.upc.edu

Sonia Rabanaque Rodríguez[†]
*Universitat Politècnica de Catalunya*
Barcelona, Spain
sonia.rabanaque@estudiantat.upc.edu

Valentina Lenarduzzi
*LUT University*
Lahti, Finland
valentina.lenarduzzi@lut.fi

Silverio Martínez-Fernández
*Universitat Politècnica de Catalunya*
Barcelona, Spain
silverio.martinez@upc.edu

***Abstract*—Developers work on different tasks in different conditions based on individual technical skills and personal habits. Identifying developer groups by mining their repositories is key for various tasks ranging from understanding developers types in open source projects, to help project managers concerned with the team allocation and coordination of human resources in companies. We aimed at identifying distinct groups of developer profiles based on well defined characteristics and at characterizing the most common quality issue types introduced by each profile in their code. We considered 77,932 commits of 33 open source Java projects, clustering their 2460 developers using dimensionality reduction techniques and applying the k-means algorithm. We identified five profiles among 2460 developers based on project experience, developer productivity and the common quality issues they introduce in the code. Results can be used by developer teams to detect and cope with harmful practices, in order to be more efficient by reducing the number of bugs they produce, looking for adequate training options, and balancing their teams.**

***Index Terms*—machine learning, developer characterisation, SonarQube, code quality, software engineering**

## I. Introduction

During the software development process, developers work on different tasks in different conditions. Since developers are human, they differ on technical skills and personal habits to face or solve code problems. Similar developer groups can be identified on factors such as time spent in each activity or quality issues introduced in the code.

Understanding the interaction between technical tasks and teams' behaviors and, consequently, the effect of their actions (e.g. software quality, productivity) can help project managers concerned with the team allocation, coordinating developers' efforts and human resources (given their current skill set and expertise) [1], especially in large, globally-distributed projects [2].

In this context, the main goal of this paper is to find distinct groups of developer profiles based on distinct characteristics. Moreover, we performed an analysis to identify the most common types of quality issues each developer profile introduces in the code.

We considered 77,932 commits from 33 Java projects included in the *Technical Debt Dataset* [3] by clustering the developers using dimensionality reduction techniques such as PCA and tSVD and applying the k-means algorithm. We identified five profiles among 2460 developers based on project experience, developers' productivity and the quality issues they introduce in the code. We refined the clustering by common characteristics into three main groups.

Results can be used by developer teams to balance their skills identifying the most common technical issues and providing a set of training options. Also, at individual level, results can be used by developers to combat some “destructive” practices and avoid behaviours that are considered harmful. Furthermore, bearing in mind that developers want to improve their abilities and managers are (or should be) interested in dealing with their projects more efficiently, we see the potential results and the adopted approach as very favourable for any company.

The main contributions of this paper are:

- The classification of open source developers into five profiles, refined into three main developer types: the

[†]Cristina Aguilera González, Laia Albors Zumel, Jesús Antoñanzas Acero and Sonia Rabanaque Rodríguez are all first authors of this paper. They equally contributed to this work and are ordered alphabetically.

cleaner, the average developer, and the dirty.

- An investigation to understand the impact of each developer profile into code quality.
- A replication package with a set of scripts that can be adopted by development teams and researchers to identify developer profiles in their context.

The rest of this paper is structured as follows: Section II describes the empirical study design. Section III presents and discusses results. Section IV discusses the potential usage of the results. Section V identifies the threats to validity of this work. Section VI reports on related work. Finally, Section VII draws the conclusion and highlights the future work.

## II. The Empirical Study

### A. Goal and Research Questions

The *goal* of the empirical study is to *analyze* developers' contributions *with the purpose of* characterizing their profile *with respect to* the quality issues they introduce in the code during the development process *from the perspective of* software engineering researchers and *in the context of* 33 open source projects.

Based on our goal, we derived two Research Questions (RQ$_s$):

**RQ$_1$.** What are the different developer profiles?
**RQ$_2$.** Which are the quality issues usually introduced in the code by each developer profile?

In **RQ$_1$**, we aim at distinguishing between different developers profiles. We implement a segmentation model that creates different profiles of developers based on the selected attributes. After having classified each developer profile, in **RQ$_2$** we investigate which quality issues each of them introduce in the code. We aim at helping developers be more efficient by reducing the number of quality issues, and to balance their teams with the needed developer types.

### B. Context

We considered the *Technical Debt Dataset*, which is a curated collection of data coming from 33 Java projects mostly pertaining to the APACHE SOFTWARE FOUNDATION ecosystem. Despite belonging to a single ecosystem, the projects of this dataset were originally selected by following the diversity guidelines introduced by Nagappan et al. [4], i.e., they were selected by addressing the representativeness of projects in terms of age, size, and domain, other than considering the Patton's "criterion sampling" [5], namely they are more than four years old, have more than 500 commits and 100 classes, and have more than 100 issues reported in their issue tracking system. As such, this dataset minimizes by design possible threats to external validity.

The *Technical Debt Dataset* includes 77,932 commits with information collected from several tools such as SonarQube 7.2, GitHub, and Jira.

As for quality issues, we considered the ones reported by SonarQube. SonarQube is a static analysis tool that includes rules for Java issues relating to different aspects of source code.

If the analyzed source code violates a coding rule, or if a metric is outside a predefined threshold (also named "gate"), SonarQube generates an "issue". SonarQube includes three types of rules: *Bug*, *Code Smell* and *Vulnerability*. It is important to note that the term "Code Smells" adopted in SonarQube does not refer to the commonly known term code smells defined by Fowler et al. [6], but to a different set of rules. The severity of each rule is ranked as an five-points scale (*Blocker*, *Critical*, *Major*, *Minor*, and *Info*).

This work has been conducted as part of the TAED2 course at UPC [7]. The first four authors composed a team in the laboratory of such course. They equally contributed to this paper. The last two authors advised the project and edited this paper.

### C. Data Collection

In this subsection, we report the data selection process, the data construction, and data integration.

*Data selection.* From the different attributes contained in the *Technical Debt Dataset*, we selected data related to the developers and commits. Regarding the tables related to issues (from Jira (*JIRA_ISSUES*) and SonarQube (*SONAR_ISSUES*)), we selected also the attributes that define both Jira and Sonar issues, that is, their type and priority/severity. In addition, we were interested in the measures obtained by Sonar to have information about the code and the reliability and security issues (*SONAR_MEASURES*).

We were also interested in the information obtained through the SZZ algorithm in order to identify fault-inducing commits (*SZZ_FAULT_INDUCING_COMMITS*). To characterize the developer's profile, it was also important to know the changes performed in each commit (*GIT_COMMITS_CHANGES*). Finally, we took the type of refactoring that has been applied to the code (*REFACTORING_MINER*).

*Data cleaning.* After the selection of the appropriate attributes, we cleaned the data by dealing with the missing values: either erasing the records that contain one or more (in each table), either relabeling them with another value. In the tables with missing values the following changes were made:

- `JIRA_ISSUES`: the records that have no value in the `priority` attribute were removed since we could not obtain this information from anywhere else. From those remaining, we relabeled the `resolutionDate` using the timestamp of the last commit of the project. In addition, in the case of the `assignee`, the missing values were relabeled as *"not-assigned"*.
- `SONAR_ISSUES`: the records with *NaNs* in `debt` were erased since we could not obtain this information. After that, the rows with `closeDate` but not `closeCommitHash` were removed. With the remaining ones, the missing values were replaced by the timestamp of the last commit of the project and the string *"not-resolved"*, respectively.

- `GIT_COMMITS`: the rows that contained a missing value in both `committer` and `author` were erased.
- `REFACTORING_MINER`: as in the previous one, the records with *NaNs* in `commitHash` and `refactoringType` were removed since this information could not be extracted from other tables.

*Data construction.* After the cleaning, we derived new attributes that we thought could be more useful for our mining goals:

1) *Developer productivity (part of $RQ_1$)*
   - Time spent in each project per developer: subtract the first `committerDate` from the last `committerDate` in table *GIT_COMMITS*.
   - Number of commits per developer (using *GIT_COMMITS*).
   - Fixed issues per developer: the number of Sonar and Jira issues, and SZZ faults that each developer has fixed in its commits, using tables *GIT_COMMITS*, *SONAR_ISSUES*, *JIRA_ISSUES*, and *SZZ_FAULT_INDUCING_COMMITS*.
   - Time to resolve Jira and Sonar issues: subtract `creationDate` from `resolutionDate`, from *JIRA_ISSUES* table; and subtract `creationDate` from `closeDate`, from *SONAR_ISSUES* table.
   - Code contributions per developer: compute, for each attribute in *SONAR_MEASURES*, the difference between their values in consecutive commits.
2) *Code Quality (part of $RQ_1$ and $RQ_2$)*
   - Induced issues per developer: the number of Sonar issues and SZZ faults that each developer has induced in its commits, using tables *GIT_COMMITS*, *SONAR_ISSUES*, and *SZZ_FAULT_INDUCING_COMMITS*.
   - For each refactoring commit, check if they have induced an issue (tables *SZZ_FAULT_INDUCING_COMMITS* and *REFACTORING_MINER*).

*Data integration.* For the Data Analysis we needed a table with information per developer. Therefore, some of the attributes had to be aggregated. Depending on the attribute, we considered averaging or adding the values. Since not all the developers appeared in every table, after their combination some attributes had missing values. These had to be replaced with values that made sense. After this, we ended up with a table with 2,460 developers and 83 attributes.

### D. Data Analysis

**Project experience.** We used table GIT_COMMIT data, as it contains information about commits performed by each developer. For each committer, we have computed the time in days he/she has spent on each project in which he at least made one commit. This period of time is computed as the difference in days between the first and last commit he made in the project. Finally, to have a single value for each developer, we have performed the average of days between the different projects. This is equivalent to knowing on average how many days each developer spends working in one project.

From table GIT_COMMIT, for each committer (equivalent to developer), we have computed the number of different projects he/she has worked on. This is done by counting the number of different *projectID* on which the developer has made at least one commit.

**Developer's productivity.** We used table JIRA_ISSUES data, as it contains information about reported Jira issues and the user who solved the bug. This person is called assignee and we are going to consider that he/she is also a developer. For each assignee, we have computed the different issues (different "creationDate") for which he/she has fixed the bug of himself or another developer, called the reporter. We must take into account that the resultant developers are going to be the ones that fixed at least one bug. The majority of developers are concentrated in the range of bugs between 1 and 9. A small part of them are spread through values higher than 10. This way, we have identified two different developer behaviours. However, we must take into account another important group: the ones that do not fix any bug. This quantity is quite higher than the quantity of developers that fixed just 1 bug.

**Clustering developer' profiles.** The nature of our problem is to create homogeneous groups of developers that share some characteristics and differ significantly from the other groups. For that reason, we performed clustering analysis over a reduced dimensionality space. We use scikit-learn's [8] clustering and decomposition modules, in particular the k-means algorithm from the former and PCA and Transposed SVD from the latter.

The adopted clustering algorithm makes some important assumptions with respect to the data: 1) The variance of the distribution of each attribute is spherical, 2) All features have the same variance, and 3) The prior probability for all the clusters are the same (i.e. each cluster has roughly the same number of observations).

We identified the more frequent issues of each developer profile. To do that, we assume that the clusters are well defined, that is, that they separate the developers clearly according to their characteristics. We have used the centroid of each cluster, that is, the representative of the developers of the same profile, to obtain this information. In other words, once we have obtained the centroids of each profile, we have searched for the most frequent issues of this representative.

**Test design.** To test our models we applied two approaches: 1) different clearly visible profiles identification, 2) homogeneous developer groups identification (and differences).

Therefore, we had one visual test, in which we plotted the observations after the dimensionality reduction to see if the clusters are sufficiently distinguishable; and another test related to the homogeneity of the profiles, that is, we can use the variance between and within the developers of the different clusters.

**Within cluster variance:** a dispersion estimation of the observations within a cluster. It is computed as the average of the distance between individual observations and the center of their cluster (i.e., centroid). So we get one value for each cluster $k$: $\sigma_{w,k}^2 = \frac{\|X_{k,n} - c_k\|}{N_k}$, where $\| \cdot \|$ is the Euclidean distance, $N_k$ is the number of observations of this cluster, $X_{k,n}$ are the observations contained in the cluster and $c_k$ is the centroid of the cluster.

Summing over all the clusters we get the total within-cluster variance $\sigma_w^2 = \sum_{k=1}^{K} \sigma_{w,k}^2$, where $K$ is the total number of clusters.

Between cluster variance: it is an estimation of the variation between all clusters. A large value can indicate clusters that are spread out, while a small value can indicate clusters that are close to each other. This metric can be computed as $\sigma_{b,k}^2 = N_k \cdot \|c_k - \bar{X}\|$, where $\| \cdot \|$ is the Euclidean distance, $N_k$ is the number of observations of the $k$-th cluster, $c_k$ is the centroid of this cluster and $\bar{X}$ is the variables mean over all the observations (not just the ones from this cluster).

Summing over all the clusters we get the total between-cluster variance $\sigma_b^2 = \sum_{k=1}^{K} \sigma_{b,k}^2$, where $K$ is the total number of clusters.

This can also be seen in the Calinski-Harabasz index, where the within- and the between-cluster variances are used to get an index that indicates us which clustering technique has the most well-defined clusters, with a large $\sigma_b^2$ and a small $\sigma_w^2$. Therefore, the clustering algorithm that produces a collection of clusters with the biggest Calinski-Harabasz index is considered the best algorithm based on this criterion. This index is defined as $CH = \frac{\sigma_b^2}{\sigma_w^2} \cdot \frac{N-K}{K-1}$, where $N$ is the total number of observations and $K$ is the total number of clusters.

Regarding the intuition that items in the same cluster should be more similar than items in different clusters, we use the Davies–Bouldin index to assess the quality of the clustering algorithms. This index can be calculated with the following formula:

$DB = \frac{1}{K} \sum_{k=1}^{K} \max_{k \neq k'} \left( \frac{\sigma_{w,k}^2 + \sigma_{w,k'}^2}{\|c_k - c_{k'}\|} \right)$,

where $K$ is the total number of clusters, $\|\cdot\|$ is the Euclidean distance, $c_x$ is the centroid of the cluster $x$ and $\sigma_{w,x}^2$ is the within-cluster variance of the cluster $x$. Based on this criterion, the clustering algorithm that produces a collection of clusters with the smallest Davies–Bouldin index is considered the best algorithm.

To evaluate our models, since it is an unsupervised problem, we studied the different profiles obtained with each model to see if they make sense to us, considering our knowledge of the topic and what we expect to obtain.

**Model Description.** The modelling phase is comprised of three separate procedures:

*Dimensionality Reduction.* The dataset extracted from the developers' profiling contained 83 distinct features. The feature space, then, was too big to perform clustering directly over it: more than 10 dimensions is generally regarded as 'too many' for clustering with K-Means because of the curse of dimensionality. Moreover, each of the features in the original space did not explain much variance (all $< 5\%$), so feature selection was not possible. The objective, then, was to extract the maximum information possible from these 83 variables and 'compress it' into as few dimensions as possible. In order to do this, classic PCA and Transposed SVD were performed, the rationale being that, with both methods, the found PCs are linearly related to the original features, allowing for a level of explainability of the cluster representatives. That is, we can tell the importance of each of the original features to each principal component. We then picked the first $n$ 'components' that explain, as an heuristic, 75% of the original data variability. We argue that 75% is enough as the other 25% is, for the most part, noise. In the case of PCA, the number of components picked is 9 and in the case of tSVD, 10.

*Clustering.* Over the $n$ PCs extracted from the previous step we independently clustered with K-Means. We performed a series of experiments to find the optimal number of clusters over the data (silhouette analysis and the Gap statistical test). Finally, with the found clusters of the data projected onto $n$ principal components, we extracted what each of them meant. This could be done due to the fact that each principal component was mostly important to a/some original feature/s (in terms of variance explained). Then, we found the characteristic in terms of the principal components of each cluster, equivalently telling us which original features characterized the cluster. Moreover, given that we know the importance of each original feature to each principal component and that we can extract the importance of each principal component for each cluster centroid, we can quantify the relation between each centroid and each original feature. We selected the developer groups obtained from PCA or tSVD by comparing both results (Section IV).

### E. Replicability and Data Availability

In order to allow our study to be replicated, we have published a replication package[1], following the template Cookiecutter Data Science, and documentation suggested on CRISP-DM.

## III. Results

In this section, we report the achieved results that are summarize in Table I. We identified five different preliminary profiles based on project experience and developer's productivity and quality issues introduced in the code. For each profile, we outline project experience (years), number of commits where they worked (*#Commits*), number of fixed issues (*#Fixed Issues*) used to answer to our RQ$_1$. Moreover, Table I includes information related to refactorings activities performed (*#Refactorings*), if developers documented their code (*Code Documentation*), the code complexity (*Code Complexity*), and the number of Code Quality Issues introduced in the code grouped by type (Code smell, Bug, and Vulnerability).

### A. Developer's Profiles Identification (RQ$_1$)

Before applying any clustering technique, we explored the GIT_COMMIT table, which has 1016 developers, and we got the following results:

[1]Replication package: https://doi.org/10.6084/m9.figshare.14980662

TABLE I
IMPACT OF THE FIVE DEVELOPER' PROFILES ON DEVELOPER'S PRODUCTIVITY AND CODE QUALITY

| Dev. type | Profile | Project experience (years) | Developer's productivity | | Code Quality | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | #Commits | #Fixed Issues | #Refactors | Code Documenting | Code Complexity | #Introduced Issues | | |
| | | | | | | | | CS | Bug | Vuln. |
| The cleaner | P1 | medium | medium | high | low | medium | medium | low | low | low |
| The average developer | P2 | low | low | low | low | medium | low | low | low | low |
| | P3 | high | high | low | medium | high | high | high | medium | medium |
| | P4 | medium | medium | medium | low | low | medium | medium | low | low |
| The dirty | P5 | high | high | low | high | low | high | high | high | high |

CS means Code Smells and Vuln. means Vulnerabilities

*Project experience*. We identified three main groups among 1016 developers:

- 363 developers who have been very few days (less than 24 days on average) working in the project. They can be considered as contributors to the project but not developers.
- 429 developers who work on projects for a reasonable period of time (less than 2 years on average).
- 224 developers who work for many days (more than 2 years on average).

Moreover, we have computed the number of different projects where a developer worked on, identifying three main groups:

- 901 developers that just worked on a single project
- 66 developers who worked in a moderate number of projects (2-3 projects)
- 49 developers that participate in lots of projects (>3 projects)

*Developer's productivity* We identified three main groups:

- Developers that just worked on one single project: on average they are used to fix 134.3 bugs.
- Developers who worked in a moderate number of projects (2-3 projects): we are not able to find any case where these developers solve any bugs.
- Developers that participate in lots of projects ($>$ 3 projects): their mean value is 191.5, which is quite superior to the first case but more or less similar.

After these, we took the dataset with 2460 developers obtained in Section II-C (combining all the tables) and applied K-Means:

*Clustering developer profiles.* tSVD reduced the data dimensionality identifying the most important attributes that characterize the profiles as follows:

**Profile 1 (1950 developers):** Developers are responsible for some commits (39 on average) and they are the ones that fix more Jira issue (636 on average).

**Profile 2 (14 developers):** Developers almost never do commits (0.44 on average). They spend a lot of time working on Jira issues (4 years on average) but rarely fix them (3 on average).

**Profile 3 (224 developers):** Developers are responsible for a fair amount of commits (309 on average) and they almost never fix Jira issues (0.46 on average).

**Profile 4 (238 developers):** Developers are responsible for some commits (15 on average) and they also fix some Jira issues (12 on average).

**Profile 5 (34 developers):** Developers are responsible of a lot of commits (2168 on average). Moreover, they induce many bugs applying refactorings (856 on average).

## *B. Code Quality Issues by Profiles (RQ$_2$)*

From the clusters obtained, we identified the most code quality issues in each profile' groups:

**Profile 1**: Developers that do not usually document their code (0.01 commented lines on average). Their code complexity is low (increase the complexity 22 points on average) and introduce some issues (3 on average), mainly Violations or Code Smells with Major or Minor priority.

**Profile 2**: Developers that do not usually document their code. The complexity of their code is very low (decrease it 16 points on average). These developers never induce issues of any kind (0.4 on average), but if they create one, it is usually of type Violation or Code Smell and Major or Minor priority.

**Profile 3**: Developers that document the most of their code (0.11 commented lines on average). The complexity of their code is high (increase the complexity 2464 points on average). These developers usually introduce Violations and Code Smells with Major or Minor priority. This is the second profile which makes the most number of issues (49 on average).

**Profile 4**: Developers that never document their code. The complexity of their code is low (increase the complexity 24 points on average). In general, this profile does not induce issues (0.44 on average), but regarding the few that they create, they are Violations and Code Smells with Major or Minor priority.

**Profile 5**: Developers that do not document their code. The complexity of their code is very high (increase the complexity 31420 points on average). These developers are the ones that induce the most number of issues in general (417 on average). The most introduced ones are Violations and Code Smells with Major or Minor priority.

## IV. DISCUSSION

After the implementation of the model with the selected data, we obtained five different developers profiles. Then, we evaluated the results according to the developers profiles that the model created.

First of all, we tried to test the results using visualization techniques. We saw that the clusters were separated in three main directions, but we could not see them clearly forming groups.

We used techniques related to the homogeneity of the profiles using the variance between and within the developers of each cluster. Comparing both variances of each technique, we saw that the values obtained were similar, but it seemed that the tSVD technique was slightly better. It was due to the fact that the within cluster variance was lower than the one obtained with PCA, and because the between cluster variance was greater. Moreover, we used some indexes that related these values and that indicated which technique had the most well-defined clusters. With the first one, the Calinski-Harabasz index, it seemed that the results were better when we used the tSVD to reduce the data dimensionality. However, using the Davies-Bouldin index, we obtained that the best results were obtained from the PCA. Finally, we remark that these results did not check if the results were good or not, but they gave us insights to select the more appropriate model.

Moreover, we saw that the conclusions obtained from the tests were not conclusive, as we could not deduce which technique was the best one. So, we decided to use the attributes that distinguish the clusters to select the best technique according to a subjective measurement. Therefore, we looked for the most important attributes that characterize each profile defined by each one of the techniques. In both cases we found profiles that were similar and that passed the tests, as we considered that the profiles were well defined and separated the developers in five profiles with different characteristics. Finally, we selected the tSVD as the best technique since the profiles seemed more consistent and their characteristics made more sense. We found two profiles with distinct characteristics, and another three that were in between these two groups and that differed on the characteristic values of the main attributes.

From these obtained profiles we got a list of their frequent issues to get the most frequent errors of each cluster. In all cases we saw that they tend to introduce the same type of issues and with the same priority. That is, all of them create, to a greater extent, Violations and Code Smells of Major or Minor priority. This means that the differences between profiles were in the amount of issues they create, not in their type.

From the five groups obtained after dimensionality reduction with tSVD, we could have studied the most important attributes that characterize the profiles and their values in each group. Profile 5 is made up of developers that make the most issues, they do not document their code, they do not fix Jira issues, their code is very complex, they make many refactors and their code is more expensive to fix than to make. On the other hand, Profile 1 corresponds to the ones that fix most of the Jira issues, rarely produce issues and their code is highly complex, even though they do not document much of their code. The other three profiles are in between these two. For example, even though Profile 2 induces very few code quality issues and its code has a low complexity, it does not fix issues; despite inducing a lot of issues and having a code with a high complexity, Profile 3 documents a lot its code; and, finally, although Profile 4 induces very few issues and its code is not very complex, it does not document much their code nor fix issues. Therefore, we think that we can join them in order to create the third profile. This way, we have obtained three separated developer profile groups as depicted in Figure 1: The *cleaner*, the *average developer*, and the *dirty*.

## V. Threats to Validity

We adopted the measures detected by SonarQube. We are aware that the SonarQube detection accuracy of some rules include false positives and negatives, and that some detected issues are duplicated due to the violation occurring in the same class and in the same position but with different resolution time. However, we replicated the same conditions adopted by practitioners when using the same tool. Unfortunately, some projects in our dataset do not tag the releases. We know that some detected issues are duplicated, since SonarQube reports the issue violated in the same class and in the same position but with different resolution times. We are aware of this fact, but we did not remove such issues since we wanted to report the results without modifying the SonarQube output.

We selected the 33 projects of the Technical Debt Dataset that projects from the Apache Software Foundation, which incubates only certain systems that follow specific and strict quality rules. Moreover, we only considered Java projects and SonarQube provides a different set of quality issues for each language.

As for conclusion validity, the obtained profiles are dependant on the cluster quality: other clustering techniques or different number of clusters could give different results.

## VI. Related Work

Some models have been proposed to identify developer's profiles [9]–[12]. However, there are a few studies that investigated developer characteristics based on artifacts (e.g., defects) in order to recommend specific tasks (e.g., bug fixing) or for mapping developer profiles.

Some models are based on developer activities [13] or technical skills [11]. Other models are constructed based on more specific information, such as issues introduced in the code [10], refactoring prioritization activities [9], effort estimation [14], or the introduced technical debt [12]. All these factors have been considered separately without combining them together in order to construct a more accurate developer profile.

Analyzing each model in depth, adopting personal information and programming skills turned out to be successful factors for code recommendation and programming task assignments [11]. Moreover, peripheral developers, active developers, core members and project leaders can be considered as other positive factors to determine developer profiles [13].

Other work built developer profiles based on the coding habits, looking at how developers were responsible for introducing and fixing static analysis violations [10]. Considering bug prioritization is also a good factor to create the model

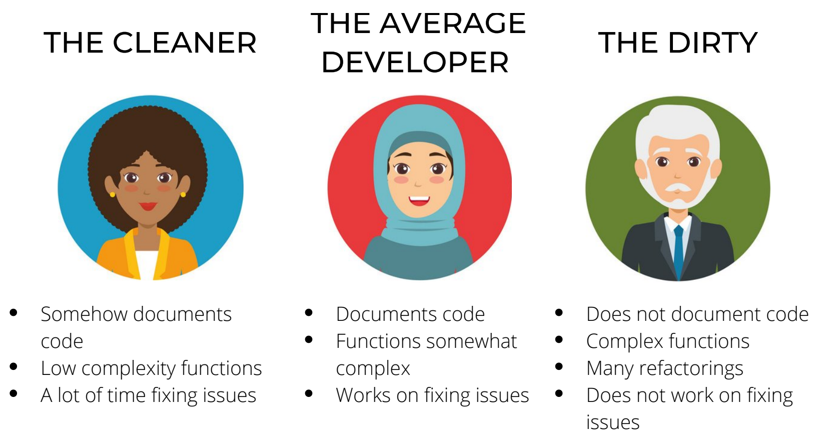

Fig. 1. Developer profiles clustered in three main groups

to more accurately assist in tasks such as bug triage, severity identification, and reopened bug prediction [9]. Results are also confirmed in another study that created the models to recommend developers for bug fixing [9].

Information from source code management systems, mailing list archives, and bug tracking systems have proven to be good factors to characterize developer activities for more accurate effort and cost estimations [14]. Profiling developers considering the accumulated technical debt has been explored evaluating code smell and refactoring perspective [12]. In particular, this model focused on developers who introduced and removed code smells and the relative refactoring actions they performed.

A similar approach was considered in another study that focused on the developers' behavior, the analysis considering technical debt. The model clustered Git commits, faults and static analysis issues (calculated by SonarQube) [15].

In contrast, in our study, we considered a combination of many factors to construct our model. We considered factors that independently have been considered harmful in the evaluated studies.

## VII. Conclusion

In this paper we characterized 33 Java projects with a total of 77,932 commits in order to identify developer profiles based on 83 distinct characteristics, using the 9 most important ones (Table I) to narrow down on the profiles, and to characterize the most common types of quality issues each developer profile introduce in the code.

Results show that there are distinct types of developer profiles. We identified five profiles among 2460 developers based on project experience, developer productivity and the common quality issues they introduce in the code. We refined the clustering by common characteristics into three main groups: The *cleaner*, the *average developer*, and the *dirty*. These results and the replication package can be used in software projects to help developers be more efficient by reducing the number of the introduced technical issues and to human resources departments to balance their teams.

As future works, we will consider the developers' personality as a possible influence on the developers' profiling [16]. Moreover, we are planning to conduct an industrial validation to evaluate the usefulness of these profiles, for instance to motivate each developer type, and to explore the personalized training they need to better contribute to the success of the software projects.

## VIII. Data Availability

We provide a replication package available on https://doi.org/10.6084/m9.figshare.14980662.

## Acknowledgment

The research presented in this paper has been developed in the context of the TAED2 course at the GCED@FIB. This work has been partially funded by the "Beatriz Galindo" Spanish Program BEAGAL18/00064 and by the DOGO4ML Spanish research project (ref. PID2020-117191RB-I00).

## References

[1] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: An empirical study on count and network metrics," in *International Conference on Software Engineering*, 2017, pp. 164–174.
[2] J. Herbsleb, "Building a socio-technical theory of coordination: Why and how (outstanding research award)," in *International Symposium on Foundations of Software Engineering*, 2016, p. 2–10.
[3] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *Conference on PREdictive Models and data analycs In Software Engineering*, 2019, pp. 2 – 11.
[4] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Joint meeting on foundations of software engineering*, 2013, pp. 466–476.
[5] M. Patton, *Qualitative Evaluation and Research Methods*. Sage, 2002.
[6] M. Fowler and K. Beck, "Refactoring: Improving the design of existing code," *Addison-Wesley Longman Publishing Co., Inc.*, 1999.
[7] S. Martínez-Fernández, C. Gómez, and X. Franch, "Aprendizaje basado en proyectos de analítica de software en estudios de ciencia e ingeniería de datos," *Actas de las Jenui*, vol. 6, pp. 27–34, 2021.
[8] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[9] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *International Conference on Software Engineering*, 2012, pp. 25–35.

[10] P. Avgustinov and other, “Tracking static analysis violations over time to capture developer characteristics,” in *International Conference on Software Engineering*, vol. 1, 2015, pp. 437–447.

[11] W. Y., M. P., Y. Z., and Z. H., “Developer portraying: A quick approach to understanding developers on oss platforms,” *Information and Software Technology*, vol. 125, p. 106336, 2020.

[12] Z. Codabux and C. Dutchyn, “Profiling developers through the lens of technical debt,” in *International Symposium on Empirical Software Engineering and Measurement*, 2020.

[13] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, “Evolution patterns of open-source software systems and communities,” in *IWPSE '02*, 2002.

[14] J. Amor, G. Robles, and J. Gonzalez-Barahona, “Effort estimation by characterizing developer activity,” in *International Workshop on Economics Driven Software Engineering Research*, 2006, p. 3–6.

[15] X. Li, “Research on software project developer behaviors with k-means clustering analysis,” in *Summer School on Software Maintenance and Evolution*, 2019.

[16] F. Calefato, F. Lanubile, and B. Vasilescu, “A large-scale, in-depth analysis of developers' personalities in the apache ecosystem,” *Information and Software Technology*, vol. 114, pp. 1 – 20, 2019.