Resource Classification from Version Control System Logs

Kushal Agrawal[‡], Michael Aschauer[*], Thomas Thonhofer[*], Nico Tomsich[†]
Saimir Bala[*], Andreas Rogge-Solti[*]
[‡]kushalagrawal1412@gmail.com,
[†]nico.tomsich@gmail.com,
[*]{firstname.lastname}@wu.ac.at
Wirtschaftsuniversität Wien
Institute for Information Business, Vienna, Austria

***Abstract*—Collaboration in business processes and projects requires a division of responsibilities among the participants. Version control systems allow us to collect profiles of the participants that hint at participants' roles in the collaborative work. The goal of this paper is to automatically classify participants into the roles they fulfill in the collaboration. Two approaches are proposed and compared in this paper. The first approach finds classes of users by applying k-means clustering to users based on attributes calculated for them. The classes identified by the clustering are then used to build a decision tree classification model. The second approach classifies individual commits based on commit messages and file types. The distribution of commit types is used for creating a decision tree classification model. The two approaches are implemented and tested against three real datasets, one from academia and two from industry. Our classification covers 86% percent of the total commits. The results are evaluated with actual role information that was manually collected from the teams responsible for the analyzed repositories.**

***Keywords*—Role discovery, Version Control Systems, Software Projects**

I. Introduction

In the field of software engineering, version control systems (VCS) such as Git or Subversion (SVN) have become an indispensable tool and are used for the majority of collaborative development projects. The repositories of these systems store data about projects and their contributors, beyond the raw source code they committed [1]. From a management perspective, it is interesting to analyze repository data for compliance purposes. From an academic perspective, repositories hold valuable information about the way people work and collaborate.

In this paper the focus lies on mining and analyzing properties of the users. More specifically, it aims at classifying the users which appear in the logs. This idea is based on the assumption that different types of users utilize the system in different ways and that these differences are reflected in their commits and subsequently in the logs. More precisely, we are interested in the following research questions:

1) Is it possible to classify resources from VCS log data?
2) To what extent can resources be assigned to classes?
3) What are the main differences among these classes?

The main objective is to find a way of classifying users automatically, using an algorithm built on common data mining and machine learning methods. The devised algorithm answers these research questions.

This paper is structured as follows: Section 2 provides the basis for this paper, links it to related work and formulates the research questions. Section 3 presents two approaches on how users can be classified in version control systems. In Section 4 the implementation of two algorithmic solutions is described, followed by an analysis of the results. Section 5 concludes the paper.

II. Background

This section introduces the problem setting, discusses related work, and details the research questions addressed in this work.

A. Problem description

The problem we address in this section is the discovery of the roles that members of a software project may actually play in a collaborative setting. Each member is assigned to at least one role in the project. Roles are decided in the project planning phase. This phase also involves the definition of project tasks and the assignment of suitable tasks to the various roles.

Projects follow clear guidelines. Guidelines may come from internal policies or from rules and regulations of the working domain. For example, software systems in the railway domain must make sure that their development process and resources comply with safety requirements imposed by the European standard EN50128. This is why project managers need to track how they distribute work to different people and whether the project members effectively contribute according to their role and their task.

Software configuration management (SCM) systems are a valuable source of information to investigate on the behavior of project members. These systems are used for tracking and controlling changes in the software. If a change produces a wrong or undesired outcome, it is always possible to revert to an older configuration of the system. Artifacts' versions are automatically managed by VCSs. It is always possible to follow the evolution of each artifact, along with information

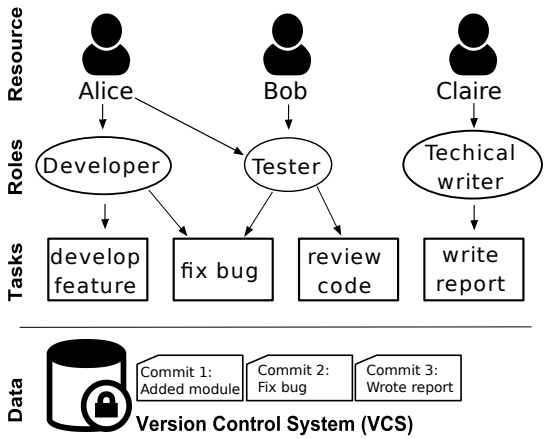


Fig. 1: Software project and resources

about the resources who changed it and their comments, by looking into the VCS logs.

Figure 1 illustrates how people work in a project. People are assigned to defined roles and contribute to shared tasks or perform tasks alone. Their contributions are part of generated artifacts. As the project is enacted, the number and the content of the artifacts in a project grows. Changes in the artifacts themselves and in the structure of the project reflect the development process that project workers follow. The evolution of the repository mirrors the progress of the project.

VCS logs provide rich and fine grained information about the changes in the project. A change may consist of a file being modified or new files being added to or removed from the repository. When changes are complete, it is possible to store them in the repository through a commit. Each commit contains a unique revision number, the identity of the resource who issued the commit, a timestamp, statistical information about the changes for each affected file, and a comment from the person who committed.

Let us consider the following example of how people use a VCS to collaborate in a project. Alice is a software engineer at Abc. ltd, and works on a project with her colleagues Bob and Claire. Alice is mainly assigned to the role *developer*, but she can also be a *tester*. Her tasks include the development of new features and fixing of related bugs. When a feature is ready she performs a commit and adds a new message where she describes her work, e.g. "added new module to demo". At the same time she updates a file named "rule". This is reflected in the VCS log, like in the first row of Table I, identified by commit id 1. Bob is a *tester* whose task is to ensure that the code submitted by Alice works properly. He discovers a bug in the "setup" interface file. Bob fixes those minor bugs and informs Alice of further work needed on the feature. Meanwhile, he commits his changes with commit id 2 and comments "Modified the setup interface". Consequently, Alice reworks her code and commits a new version as reported in row 3 of Table I, commenting her work with "Update application interface". As a *technical writer*, Claire takes over and starts to work on the documentation. She commits part of her work as in row 4. As the project continues, the work is accordingly stored in the log as in Table I.

As we can see from the example, user work is tracked in a fine granular way by the VCS. The challenge is to use log information for getting insights into the roles of project members. The following section discusses related literature that has addressed role discovery.

B. Related Work

Role discovery has been addressed by literature in different settings and from several points of view. Here we highlight existing efforts from a data perspective.

1) Structured data approaches: This class of methods includes algorithms that make use of quantifiable data. We divide them into: *a)* mining software repositories (MSR) approaches; and *b)* process mining (PM) approaches.

a) Mining software repositories approaches: In the area of MSR, Liguo and Ramaswamy [1] use a hierarchical clustering based on user interactions to identify two categories of users: *core member* and *associate member*. Core members are those users whose interaction frequency is higher than a given threshold. Associate members are instead users whose interaction frequency is below the threshold. Alonso et al. [2] use a rule-based classifier that maps file types onto categories and hence each author who modified a file is linked to the files' category. Gousios et al. [3] classify developers contribution based on lines of code (LOC) changes and infer activities from them. Begel et al. [4] developed the Codebook software tool a utility for finding experts. They use a social network approach that combines sources from people, artifacts, and textual references to other people. Ying and Robillard [5] study developer profiles in terms of their interaction with the software artifacts to understand how they modify files and to further recommend changes based on history from VCS logs. Füller et al. [6] investigate user roles in innovation-contest communities. They use quantitative methods to analyze user activity logs and interpretative to categorize qualitative comments into classes.

b) Process mining approaches: Efforts have been done to analyze software repositories with process mining techniques. Rubin et al. [7] implement a multi-perspective incremental mining that is able to continuously integrate sources of evidence and improve the software engineering process as the user interacts with the documents in the repository. Their approach allows for mining other perspectives, such as roles, by applying social network analysis. However, only statistical methods can be applied to their output, since it lacks the comments that are associated to file changes. In the same setting, Poncin et al. [8] developed FRASR, a framework for analyzing software repositories. FRASR can be used in order to transform VCS logs data into the XES [9] data format that can be further analyzed with process mining tools like ProM[1]. Song and van der Aalst [10] focus on three types of organizational mining *i)* organizational model mining,

[1]http://www.promtools.org/doku.php

TABLE I: Example of a VCS log.

Commit ID	User ID	TimeStamp	Updated Files	Comment
1	Alice	2014-10-12 13:29:09	Demo.java rule.txt	Added new module to demo and updated rules
2	Bob	2014-11-01 18:16:52	Setup.exe	Modified the setup interface
3	Alice	2015-06-14 09:13:14	Demo.java	Update the application interface
4	Claire	2015-07-12 15:05:43	graph.svg todo.doc	Define initial process diagram & listed remaining tasks
...	...	...	...	...

ii) social network analysis, and *iii)* information flows between organizational entities. Schönig et al. [11] propose a mining technique to discover resource-aware declarative processes.

2) Unstructured data: Maalej and Happel [12] use natural language processing (NLP) for automating descriptions of work sessions by analyzing developers' informal text notes about their tasks. Developers are then classified into two classes based on their behavior: developers who use problem information to refer to their current activity and developers who refer to task and requirements. Kouters et al. [13] developed an identity merging algorithm based on Latent Semantic Analysis (LSA) to disambiguate user emails. Licorish and MacDonell [14] mined developer comments to understand their attitudes.

3) Other related work: The term *role mining* often points to role mining algorithms based on role-based-access-control (RBAC) systems. These algorithms take as input predefined roles that are given as a matrix, where each user is assigned to access permissions. A number of algorithms have been developed to mine roles from RBAC systems alone ([15], [16]) or combining their data with process history logs, as in Baumgraß et al. [17]. A survey of existing techniques and algorithms can be found in [18]. Our work is disjoint from this class of algorithms as VCS does not contain access control information. Bhattacharya et al. [19] propose a contributor graph-based model. By constructing a source-based profile, and a bug-based profile, they are able to identify seven roles: *patch tester*, *assist*, *triager*, *bug analyst*, *core developer*, *bug fixer*, and *patch-quality improver*. Hoda et al. [20] use a grounded theory (GT) approach to study agile teams. Their work distinguishes the roles of *mentor*, *coordinator*, *translator*, *champion*, *promoter*, and *terminator*.

This work builds upon existing literature in that it strives to get insights on organizational level like in [7] and [10], but it takes into account unstructured data. Differently from the literature that works with unstructured data, we explicitly consider the problem of role discovery, i.e. label resources with roles. Lastly, this approach differs from [19] and [20] since we further adopt NLP techniques.

C. Research Questions

As mentioned in the previous section, the focus of this paper lies on mining and analyzing properties of the users of VCS. Users belong to preset classes and they commit message styles. Based on this assumption, the following research questions are defined.

RQ1 *Is it possible to classify resources from VCS log data?* We want to scrutinize whether a classification of the users of a VCS is possible by using the information contained in the log files. Comment messages and other data must be investigated for useful features. The classification must separate the users into clusters, and the most expressive features and their combinations must be identified. These clusters can then be further analyzed by means of the next research question.

RQ2 *To what extent can resources be assigned to classes?* Based on the created clusters, meaningful classes should be derived. The chosen features influence the types of classes that can be created. These classes must present an intelligible distinction among each other. This research question also inspects how detailed this distinction can become. Moreover, it aims at creating profiles for the class members reflecting behavior, based on the analyzed features.

RQ3 *What are the main differences among these classes?* The last research question examines the differences among the created classes in detail. It compares the results from different log files and identifies the reasons for dissimilarities. Therefore, it evaluates the quality of the classification and its applicability for different version control systems.

III. Solution Concept

Commits contain fine grained information about many aspects of a work unit. We are interested in three aspects. First we consider distinct identities and timestamps of commits. This makes it possible to compute how frequent and how distant in time commits appear in a project. Second, we obtain information about the authors. Authors are explicitly stated in the commit. Third, we consider their comments. Comments are written in natural language and need to be parsed and preprocessed before we can use them.

Once data is preprocessed, the analysis can start with a direct approach of connecting one commit message to one user and to the corresponding role. In this approach user roles are linked to the authors of commits. However, roles are not always available a priori. In these cases, a commit-based approach can be adopted. This method operates on the commits and assigns types to them first. Further analysis then maps these types to possible user roles. This approach is not a direct mapping. Instead, it allows for more flexible role

discovery. We use this approach to also predict roles, based on user profiles that we create from the commit types.

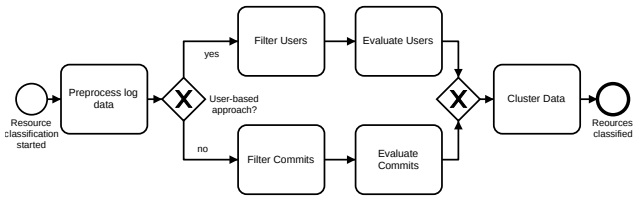


Fig. 2: Approach to role classification from VCS logs

Figure 2 illustrates the steps of our approach through a BPMN diagram. In the first step we preprocess the data and parse the SVN log file. Then we account for two different types of classification: user-based and commit-based. These approaches require a prior step where we filter the data according to users or to commits. Both approaches include an evaluation step where we map keywords and information on file types to users or commits, respectively. The last step is the data classification. Section IV describes both approaches in detail.

IV. Implementation

We implemented the outlined solution concept in a Python script. This script automatically fetches, processes and analyses the log files and creates a classification model that is based on the extracted information. The following tools are used for implementation:

Technology. Python is a general-purpose, high-level programming language. It is open source, easy to use and offers various third party modules[2].

Machine Learning. The Scikit Learn module is used for machine learning[3]. We used decision trees (DTs) for classification and regression. DTs are a supervised learning method whose goal is to create a model that predicts a target feature of variable by inferring existing rules from the data.

Natural Language Processing. The Natural Language Toolkit (NTLK) offers methods to extract additional information from everyday communication. Among others, we rely on the *bag of words* technique that analyzes word occurrences[4].

Next, we discuss the data selection criteria. While there are many accessible code repositories, picking a dataset of appropriate size and quality proves to be rather challenging. We need to consider:

Diversity. Classification on a single repository might not produce a generally applicable result, whereas using more repositories increases complexity.

Size. Individual repositories' size is determined by the number of commits.

[2]https://www.python.org/

[3]http://scikit-learn.org/stable/

[4]http://www.nltk.org/book/ch00.html

Roles. The real roles of the contributors of a repository is required for certain classification steps and the verification of results.

Type. Differences in organizational aspects of the projects: the type of the development team (e.g. professional or hobby), type of software (e.g. proprietary or open-source), and type of platform (e.g. private server or public repository hosting service) can influence the way repositories are used.

Backend. There exists a plethora of version control systems, each with their unique characteristics and formats. The approach should be able to handle popular systems including Subversion (SVN), Git, and Mercurial.

Based on these criteria, we selected three repositories for analysis, each with thousands of commits:

1) Main code repository of the company Infinica. Proprietary software. VCS: Mercurial
2) ProM Sourceforge project repository. Open source software. VCS: SVN
3) Camunda GitHub project repository. Open source software. VCS: Git

These repositories cover a broad range of the above mentioned differences. The necessary role information was reviewed by interviews with the main contributors of these projects. While two repositories were openly available, the third was granted a direct access from within the company.

An initial screening of commit messages in the repositories revealed that there is useful information within most of the messages, but the variance regarding the style and content of the messages is high between commits and users. From the messages, certain words and phrases can be extracted, which can be linked to a specific class. In a next step, the log files were preprocessed removing commits and/or users with faulty information, merging users with several accounts and anonymizing the data if required.

To unify the different systems, we create a shared object model to store the relevant information. This model consists of a commit object for each commit in the log, consisting of an id, an author, a message, a timestamp and lists of all added, modified and deleted files. From this model it is possible to derive a second model consisting of the users, by aggregating the information for each author name occurring in the commits.

In the following, we present two ways to tackle the problem of classification of users into their respective roles. First, the *user-based* approach focuses on the users at an aggregated level. Second, the *commit-based* approach classifies individual commits to allow for a finer grained classification.

A. User-based Approach

The main idea of this approach is to use clustering in order to find potential user classes and build a classification model based on those clusters, which represent a comprehensible, existing class of users. As a clustering method we used k-means.

The first step is feature selection. We identified the following features per user.

- Total number of commits.
- Timeframe: the time between the first and last commit of the user (approximates the time a user has been working on a project).
- Commit frequency: total number of commits divided by the time frame. Represents the number of commits a user makes within a certain period of time (e.g. day, month).
- Commit message length: average length of commit messages in number of words.
- Keyword occurrence: how often a certain word (e.g. "test", "fix") is used relative to the total number of words.
- Number of added/modified/deleted files.
- Affected file types: how often a file with a certain format (e.g. .java, .html) are modified by a user, relative to the total number of modified files.

From the clusters generated with this method, classification models are built using the decision tree method. This method uses tree graphs for representing the model. Each branching in the tree represents one decision based on the characteristics of a certain data point. Each leaf of the tree stands for a class. The classification is done by going through the tree for each data point and assigning it the class of the leaf it reaches. The models are trained for the three data sets individually. For verification of their quality, the models are cross validated with the other data sets respectively.

B. Commit-based Approach

There are multiple reasons for exploring a second approach in addition to the user clustering. As already mentioned above, the research led to the conclusion that in many cases a user can have multiple roles or execute many tasks not belonging to her primary role. This kind of use case is hard to cover using only the simple clustering method. Another reason is that a lot of information is lost when aggregating the commit information for users. This leads to the idea of classifying individual commits.

The algorithm iterates over commits and tries to assign types to them. The types are assigned based on the analysis of the commit message, and on the file extensions. The commit message is searched for certain keywords and phrases which are connected to types. The file extensions are searched for known file types fitting to a commit type. There are certain overlaps between commit types, for example the type addition can be a development or test commit. In those cases where one identified type is a more specific description for another, only the more specific one is included into the further analysis. The identified commits and the related keywords and file types are listed in Table II.

After assigning the commit types, the commits can be aggregated for the individual users resulting in the absolute occurrence numbers of each type for each user. Dividing each of these values by the total number of commits of the user, we get percentages for each type. These percentages tell how the work of a user is distributed among the different kinds of tasks which appear in a VCS. These user profiles are a form of classification, which is not as simple and concise as assigning one definite class for each user, but has the advantage of covering secondary roles and minor tasks. They can be useful for analyzing smaller project teams with multiple roles for one user.

TABLE II: Keyword lemmas used for classification

Class	Keywords
Test	test testcase test case unittest unit test integrationtest fitnesstest fitness test
Development	implement improve update script
Web	web spring http http rest html css servlet
Backend	engine
Maintenance	bugfix fix patch cleanup clean up clean
Refactor	refactor rename move revert
Documentation	document javadoc readme userguide guide tutorial faq translate doc i18n *.txt *.doc *.docx *.text *.tex *.pdf
Design	style icon font layout *.png *.svg *.jpg
Build	build compile release
Data	data database sql postgre postgresql
Tool	tool library framework depend upgrade
Addition	add new create
Removal	delete remove
vcsManagement	svn git mercurial
Automated	automated release, automated nightly
Merge	merge

The classification task is done on user profiles. A table of user profiles is created and a role for each user manually inserted, based on the role information of the data sets. For this part the Infinica data set is used, because it is the one with the most extensive information base and it has the most diverse and comprehensive set of roles among the shosen repositories. Four roles are assigned to the Infinica users: Web developers, other developers, testers and support. The last one is an aggregation of users who have different official roles but contribute in the VCS mainly in form of minor, supportive tasks.

The classification task was done in two ways: 1. Manual analysis of the table and derivation of rules by looking for similarities between users with the same role. 2. Automated classification in line with our original concept. For the latter the decision tree method is used. In this case the commit type percentages are used as features and the manually assigned roles as classes. The resulting decision tree model was validated against the ProM and Camunda data sets.

The final algorithm including retrieval, processing and analysis of data as well as classification and verification, consists of roughly 700 lines of code. The results for the two approaches are discussed below.

C. Results

In the first step of the user-based approach, the most expressive features of the Infinica data set were identified. The solution was tested on the Infinica data set due to the immediate availability of detailed information about the log file. The best results were achieved with the combination of the commit frequency and the occurrence numbers of test related keywords.

The commit frequency is a strong indicator of developers. Moreover, it also differentiates between the working preferences of the developers. The group with the lower frequencies tends to work locally on their machines and pushes their work when it is finished. Whereas, the developer group frequently pushes incremental changes to the repository, so that everybody is updated and works with the latest code.

Test-related keywords can not only identify testers, but also differentiate between their expertise. The sum of the words ”test”, ”tested”, ”testing”, ”tests” suffices to make a distinction between testers and developers. Additionally, manual testers use these words less often than their technical counterparts, who are more focused on automation.

The first part of the commit-based algorithm implementation, i.e., the commit classification, was successfully tested on all three data sets combined. For all three sets we achieved a similar coverage (percentage of commits which could be assigned some type). For Infinica the coverage was 88,94%, for Camunda it was 90,25% and for ProM 86,65%.

While the user profiles were originally intended as an intermediary step and a means to verify and improve the quality of our approach, they proved to be a useful perspective on user roles, beyond the simple categorisation using a single class. Especially when using a graphical representation such as the pie charts which can be seen in Figure 3a and Figure 3b, the commit distribution provides an interesting insight in the actual roles and tasks of users.

The Infinica dataset is divided successfully into expressive classes with k-means clustering with a k=4 shown in Figure 4. The developers (light grey circle and white square) are split off based on their higher frequency in the first step of the decision tree and the following classes are derived:

- **Developers - frequent committers:** The square cluster is comprised of developers which push every change to the repository.
- **Developers - heavy commits:** The circle cluster contains developers who work on their local machine and push less frequently. However, they have bigger commits containing more changes.
- **Testers - technical:** Testers focused on automating the testing process are in the diamond cluster. They are solely senior developers but not all of them are situated in this group.
- **Testers - non technical:** The triangle cluster is comprised of less technical testers which tend to do more manual testing. Additionally, it includes the professional services team.

Table III shows an excerpt of the user profiles and role assignments for the Infinica data set. The commit types development, backend, maintenance and refactor have been merged to a single type development, as the other, more specific types provided no additional value in this case. Also the types ”addition”, ”removal” and ”merge” have been omitted due to a lack of semantic meaning. The data visible in the table was used for the manual classification as well as the creation of the decision tree model.

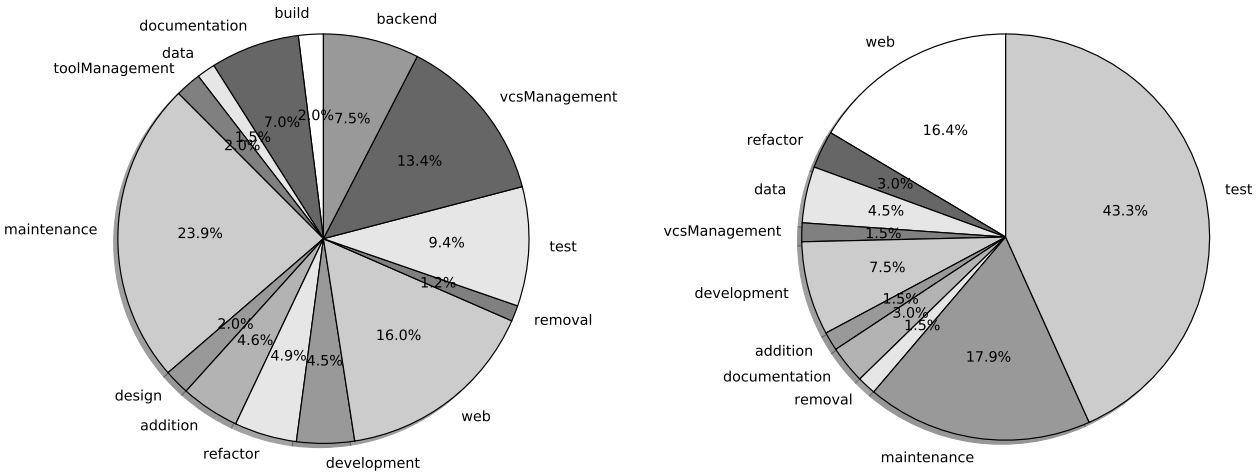
We identified 4 role-based classes which were visible from the Infinica data. Those were testers, with more than 20% of their commits being of type test and between 30% and 50% of type development, developers with above 50% development commits, web developers with more than 40% of type web, and 20% of other development and non-technical users with less than 20% development. In addition to those, we found some less obvious hints for additional classes, represented by minor, secondary roles of users. These were not represented as actual existing roles in our data, so we could not verify our assumptions. The corresponding classes we suggest for those are technical writer with more than 40% documentation commits, designers with above 30% design commits, users with various administration tasks, represented by high numbers of the types build, vcsManagment and toolManagement and database experts with a large percentage of data commits.

When applying the optimal features and clusters deduced from the Infinica data set on the ProM data set, different but still meaningful user classes are derived. This is due to the fact that the Infinica data set represents the development process within a company, whereas ProM is an academic project where researchers continuously join and leave. However, it still required to get invited for working on the project which creates an entry barrier. All members are developers without any designated testers. The ProM data set can be divided into the following four classes as shown in Figure 5.

- **Core developers:** The employed users of the ProM project are situated in the square cluster. Their commit frequency and absolute number of commits is far above the others. There is also a system user in this class. Its main task is building the project.
- **Engaged developers:** The circle cluster contains developers which also write tests. They are more committed and they put more effort into the development.
- **One-time developers:** The enagement of this group ends with the addition of their required functionality. The majority of the users falls into this class and it is represented by the diamond cluster.
- **Testers:** Despite the lack of testers in the ProM data set two have been identified as such in the triangle cluster.

The Camunda data set was analyzed based on the optimal features and classes derived of the Infinica data set. The majority of the users are developers like in the ProM data set. However, it is an open source project and users can commit anything at any time without restrictions. There is a permanently appointed core team which evaluates, selects and integrates commits for the product. The clustering creates similar classes like in the ProM data set and it can be divided into the following four classes as shown in Figure 6.

- **Core developers:** The square cluster contains the employed users of the Camunda project. Their commit frequency and absolute number commits is far above the others.
- **Engaged developers:** Developers which stay longer with the project are situated in the circle cluster. They are



(a) User profile for a backend developer

(b) User profile for a tester

Fig. 3: Commit distributions of the Infinica dataset

TABLE III: Excerpt from user profiles with roles for Infinica data set

test %	development %	web %	documentation %	vcsManagement %	build %	toolManagement %	data %	design %	class name
6.07	39.49	26.40	7.94	0.23	3.27	0.70	0.23	11.21	dev
9.41	40.90	15.99	7.00	13.37	1.96	2.04	1.46	2.00	dev
3.83	26.78	44.70	2.90	4.70	3.93	0.60	2.95	6.01	webdev
0.00	28.07	52.28	2.46	0.00	5.26	1.40	0.35	8.42	webdev
43.28	28.36	16.42	2.99	1.49	0.00	0.00	4.48	0.00	tester
23.99	27.73	15.26	7.79	0.00	2.49	1.56	1.25	4.67	tester
0.00	7.94	38.10	0.00	38.10	1.59	4.76	0.00	9.52	support
1.19	3.39	46.15	1.61	46.15	0.08	0.17	1.19	0.08	support
...	...	...	...	...	...	...	...	...	...

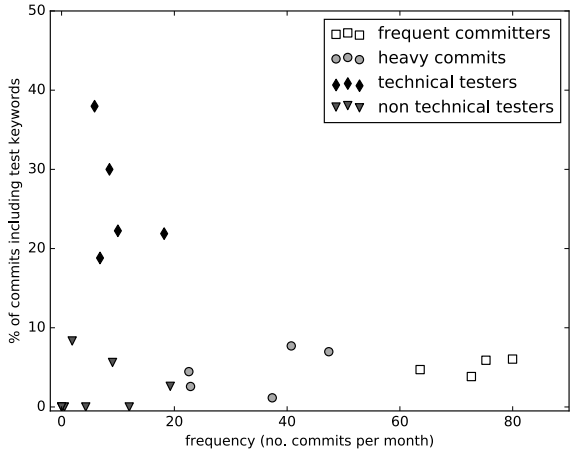


Fig. 4: Scatterplot with highlighted clusters (k=4) based on the Infinica data set.

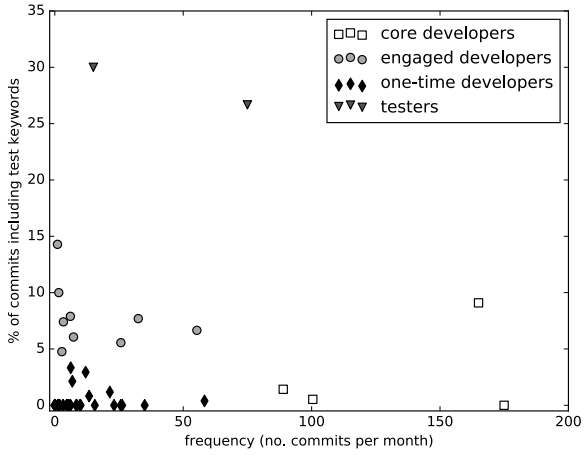


Fig. 5: Scatterplot with color-coded clusters, ProM data set

refining their own committed code or they are extending the product in different areas.

- **One-time developers:** Similar to the ProM data set the majority of the users belongs to this group and it is represented by the diamond cluster. Code usefull or not is committed typically once and there is no lasting commitment.

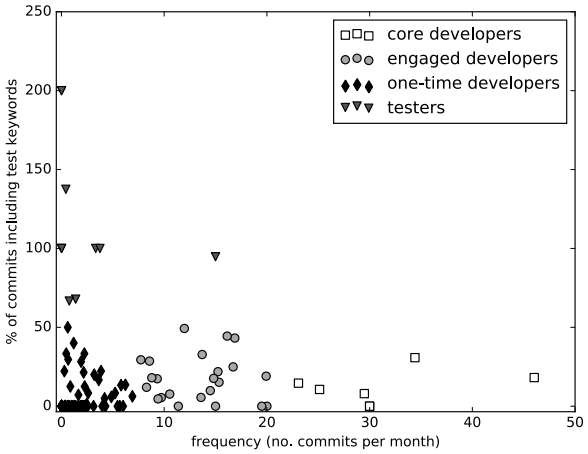


Fig. 6: Scatterplot with colorcoded clusters, Camunda data set

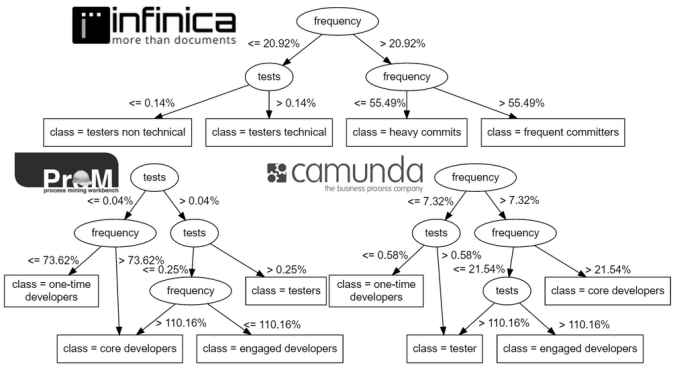


Fig. 7: Decision trees, all three data sets

- **Testers:** Also testers have been identified in the triangle cluster. They already have a lower frequency than developers and therefore it is difficult to make an estimation about their commitment.

For the user-based approach no further distinction can be made based on expertise, time in the company, teamwork, development area, project membership, room allocation, etc. In the case of a development from junior to a senior role in the course of the coverage of the log file the respective persons stay within their previous clusters. The associated increase or decrease in commit frequency can not be linked to the development.

Decision trees are an effective tool to display the formation and the partitioning of clusters in a tree-like model. The corresponding decision trees for the previous presented k-means clustering shown in Figure 7 display the boundaries of the clusters. Due to the differences in the structure of the projects, business vs open source, the data sets share only little similarities.

The Infinca decision tree initially splits the developers and the testers apart based on the commit frequency with a border value of 20.92%. The developers are split again into sublcasses

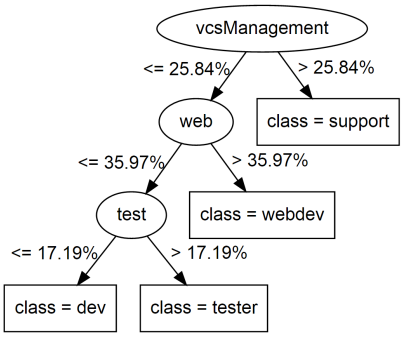


Fig. 8: Decision tree, commit based model

based on the commit frequency, which now seperates them at 55.49% into frequent committers and heavy commits. The testers are split based on test related word occurances at 0.14%, where the non technical testers use these words less often than their technical counterparts. On the contrary, the other two trees start separating the subsets right away in different orders without displaying a clean seperation between developers and testers from the beginning.

The decision tree model for the commit-based approach is depicted in Figure 8. When comparing its rules with those of our manual classification there are some clear parallels. In the tree the differentiation between web developers and other developers is done with a border value of 36% for web commits, close to the 40% threshold of the manual table. Testers are identified having more than 17% test commits, similar to the 20% boundary. The decision tree separates the support users from the rest using the vcsManagement type. Looking at the data, this rule is definitely valid for the Infinica data, however as we were not able to provide a definite explanation for this class having higher percentages for that particular type, we can not say if this rule would also apply to other data sets. One possible explanation is that all users have a similar number of vcsManagement commits within a given timeframe, but because the support role has on average a significantly lower number of commits, they account for a higher percentage in the distribution, however this is only an assumption.

In general the decision tree is more precise, but our manually created rules capture more factors of differentiation and we were able to identify some potential classes which could not be included in the programmatic classification with reasonable effort. A clear advantage of the automated model is that it can be efficiently applied to other data sets. Applying it to the Camunda and ProM data sets provided some verification for our decision tree model. We only included users with five or more commits into the cross validation. For the Prom data set which consisted of 42 users, 35 (83%) were classified as developers. This is not surprising as we knew before that the contributors of open source projects are mostly developers, which was also confirmed by our contact persons for ProM and Camunda. Six (14%) were regarded as testers, which fits

to our results in the user-based approach. One remaining user (2%) was classified as a web developer. As the ProM project has no web component, it makes sense that there are no web developers found by our algorithm. The one we found has only five commits, two of them web commits so this potential misclassification provides no evidence against the quality of the overall approach.

For the Camunda data the case is similar. Out of 66 total users, 49 (74%) percent were regarded as developers, the high number is again explained by the type of project and was confirmed by the contact person we interviewed. 10 (15%) testers were found, also mostly fitting to our knowledge of the data. Contrary to ProM, there is a web part in the analyzed Camunda project reflected by the fact that our algorithm found 7 (11%) web developers. In neither of the two validation data sets a support user was found. This might be due to the differences in the organisational structure between Infinica and the other two or it could stem from the classification rule based on the vcsManagement commit type, which we already discussed as potential source of error. All users in the Camunda and ProM data have below 15 percent vcsManagement commits, too small to fit the support class generated by the decision tree model. However there are other commit types with high percentages, such as build or toolManagement, in Camunda and Prom, which might fit the support class or a similar one.

When validating the model with the Infinica data set and then predicting the ProM and Campunda data set, only moderate results are achieved with the user-based approach as shown in Figure 9. For the Infinica data set meaningful classes can not be conveyed to the other data sets due to the structural differences of the projects.

For ProM (Figure 9 left) the trained classes are less representative than the ones created when clustering on its own, especially since the ProM data set does not include any designated testers. The prediction of Camunda (Figure 9 right) on the other hand performs better. However, the core developer class now also includes very committed contributors.

Because of the missing designated testers in both test data sets, the initial differentiation between technical and non-technical testers is not possible. Due to the moderate results of user based approach the commit based approach was initiated.

D. Discussion

Throughout this project we discovered a number of research directions to follow, methods to use and features to analyze, which made our research much more exploratory than originally planned. For our research we picked from a vast selection of potential methods and tools, a small set which seemed promising to us. The user-based approach, which was our initial plan, provided a useful insight into the data but fell short of providing generally applicable results in terms of a classification algorithm. The commit-based approach seems more valuable to us, due to its results and the much larger spectrum of information it delivers for analyzing users. The commit classification method seems quite robust and could with some extensions and refinements become a useful tool for analyzing arbitrary VCS repositories. The user profiles created based on this classification are definitely an interesting source of information and building a classification model on top of them has proven to be a legitimate approach.

Our first research question (RQ1) can be answered with: Yes, it is possible to classify users based on the information usually found in VCS logs. Our results show that at least for some typical roles in software development there can be enough information in commit messages and file types, to make certain statements about the users who created them. However this is not true for all users, especially when their commit styles and frequencies are very different.

The other two questions are more difficult to answer. The classes to be found can differ between repositories. From our experience, developers can be distinguished from other roles quite easily due to high numbers of commits and/or modified files as well as the usage of certain words and phrases. Testers can also be set apart in most cases, mainly through key words. Looking at the file types of added, modified and deleted files also reveals users in the field of web development. Beyond that there are certainly more differences and classes to be found but they are less obvious and require more research.

What do these results mean for the practical use of resource classification using VCS logs as discussed in Section II-A? One clear finding of our work is that any project team trying to use this kind of classification, should specify and enforce some guidelines for how VCS should be used. They should at least make the usage of commit messages mandatory, ideally also define a structure and vocabulary for them. While the models we created are probably not applicable for an arbitrary team or project, a team could follow our approach to create their own individual model using specific knowledge. For a more generally applicable classification model, additional research and improvements to our solution would be necessary.

1) Limitations: While our algorithms and models work well for describing our three data sets, it still needs to be tested for other repositories, especially from other domains. This would require testing with more data sets, more verification information and more manual analysis. While fetching more VCS repositories and running them through our algorithms could be done in a short amount of time, obtaining the necessary role information is challenging and the manual analysis time consuming.

A current limitation of analyzing VCS logs is that commit messages are used differently between users, repositories and teams. We have to assume that it is challenging to find one classification model that produces valid results for any VCS repository, because there are ambiguities in what certain statements mean. In the worst case, users omit commit messages, which renders classification based on text impossible.

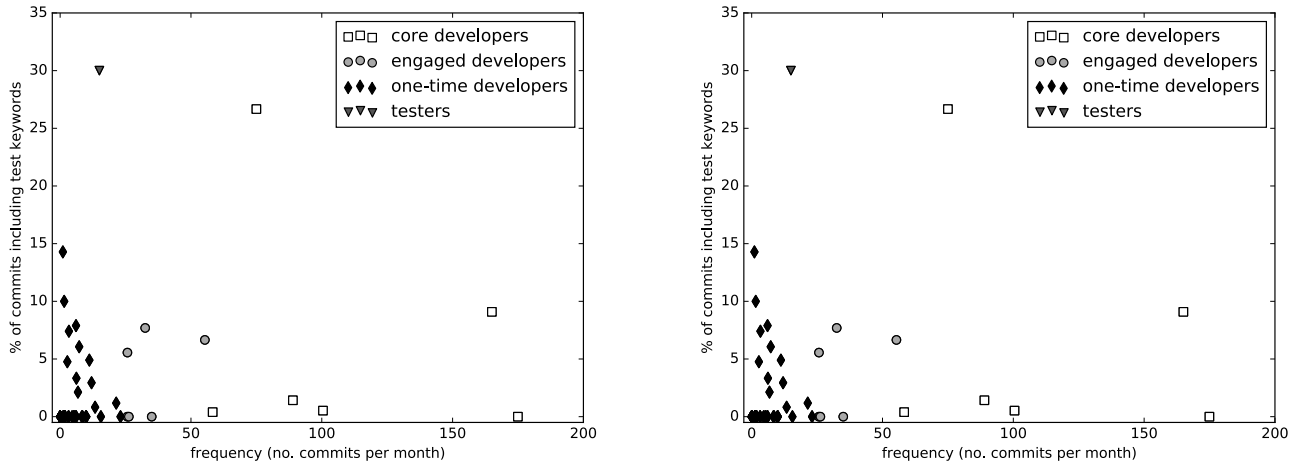


Fig. 9: Classication based on Infinica training set: Prom, Camunda

V. Conclusion and Future Work

We presented a first approach to resource classification from VCS and demonstrated feasibility of this non-trivial task. The data saved in log files contains much useful information. But in particular the structure of the underlying team and project has a big influence on finding classes and their identified attributes. Further, the many differences between individual repositories make it difficult to generalize findings.

Automating the classification with machine learning is a viable method. Even though not all required steps of creating the classification models can be done automatically and further information about the users and structure is essential. Despite current challenges, the users can be classified to a certain extent. A more precise classification, however, requires future research. One direction is to further explore clustering and classification techniques to find better suited techniques. Finally, an extension of the approach with state-of-the-art semantic reasoning seems a promising vein of research to the authors.

Acknowledgment

This work has been funded by the Austrian Research Promotion Agency (FFG) under grant 845638 (SHAPE) and the European Union's Seventh Framework Programme under grant 612052 (SERAMIS).

References

[1] L. Yu and S. Ramaswamy, "Mining CVS repositories to understand open-source project developer roles," in *Proc. - ICSE 2007 Work. Fourth Int. Work. Min. Softw. Repos. MSR 2007*, 2007, pp. 7–10.

[2] O. Alonso, P. T. Devanbu, and M. Gertz, "Expertise identification and visualization from CVS," *Proc. 2008 Int. Work. Min. Softw. Repos. - MSR '08*, p. 125, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1370750.1370780

[3] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *Proc. 2008 Int. Work. Conf. Min. Softw. Repos.* ACM, 2008, pp. 129–132.

[4] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in *2010 ACM/IEEE 32nd Int. Conf. Softw. Eng.*, vol. 1, 2010, pp. 125–134.

[5] A. T. T. Ying and M. P. Robillard, "Developer profiles for recommendation systems," *Recomm. Syst. Softw. Eng.*, pp. 199–222, 2014.

[6] J. Füller, K. Hutter, J. Hautz, and K. Matzler, "User Roles and Contributions in Innovation-Contest Communities." *J. Manag. Inf. Syst.*, vol. 31, no. 1, pp. 273–308, 2014.

[7] V. A. Rubin, C. W. Günther, W. M. P. Van Der Aalst, E. Kindler, B. F. Van Dongen, and W. Schäfer, "Process mining framework for software processes," in *Softw. Process Dyn. Agil.* Springer, 2007, vol. 4470, pp. 169–181.

[8] W. Poncin, A. Serebrenik, and M. van den Brand, "Process Mining Software Repositories," *2011 15th Eur. Conf. Softw. Maint. Reengineering*, pp. 5–14, 2011.

[9] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. Van Dongen, and W. M. P. Van Der Aalst, "Xes, xesame, and prom 6," in *Inf. Syst. Evol.* Springer, 2010, pp. 60–75.

[10] M. Song and W. M. P. van der Aalst, "Towards comprehensive support for organizational mining," *Decis. Support Syst.*, vol. 46, no. 1, pp. 300–317, 2008.

[11] S. Schönig, C. Cabanillas, S. Jablonski, and J. Mendling, "Mining the Organisational Perspective in Agile Business Processes," in *BPMDS*, 2015, pp. 37–52.

[12] W. Maalej and H.-J. Happel, "Can Development Work Describe Itself?" *7th IEEE Work. Conf. Min. Softw. Repos. (MSR 2010)*, pp. 191–200, 2010.

[13] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. Van Den Brand, "Who's who in Gnome: Using LSA to merge software repository identities," pp. 592–595, 2012.

[14] S. A. Licorish and S. G. MacDonell, "Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study," *Inf. Softw. Technol.*, vol. 56, no. 12, pp. 1578–1596, 2014.

[15] H. Lu, Y. Hong, Y. Yang, L. Duan, and N. Badar, "Towards user-oriented RBAC model," *J. Comput. Secur.*, vol. 23, no. 1, pp. 107–129, 2015.

[16] M. Frank, J. M. Buhman, and D. Basin, "Role mining with probabilistic models," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 4, p. 15, 2013.

[17] A. Baumgrass, S. Schefer-Wenzl, and M. Strembeck, "Deriving process-related rbac models from process execution histories," in *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*. IEEE, 2012, pp. 421–426.

[18] B. Mitra, S. Sural, J. Vaidya, and V. Atluri, "A Survey of Role Mining," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–37, 2016.

[19] P. Bhattacharya, I. Neamtiu, and M. Faloutsos, "Determining Developers' Expertise and Role: A Graph Hierarchy-Based Approach," *2014 IEEE Int. Conf. Softw. Maint. Evol.*, pp. 11–20, 2014.

[20] R. Hoda, J. Noble, and S. Marshall, "Self-organizing roles on agile software development teams," *Softw. Eng. IEEE Trans.*, vol. 39, no. 3, pp. 422–444, 2013.