Combined method for mining and extracting processes, related events and compliance rules from unstructured data

Deliverable D3.3

FFG – IKT der Zukunft
SHAPE Project
2014 – 845638

Table 1 Document Information

Project acronym:	SHAPE
Project full title:	Safety-critical Human- & dAta-centric Process management in Engineering projects
Work package:	3
Document number:	3.3
Document title:	Combined method for mining and extracting processes, related events and compliance rules from unstructured data
Version:	1
Delivery date:	01 April 2016 (M3)
Actual publication date:	———————
Dissemination level:	Public
Nature:	Report
Editor(s) / lead beneficiary:	WU Vienna
Author(s):	Cristina Cabanillas, Saimir Bala, Jan Mendling, Axel Polleres
Reviewer(s):	Claudio Di Ciccio, Jan Mendling

Contents

1 Introduction

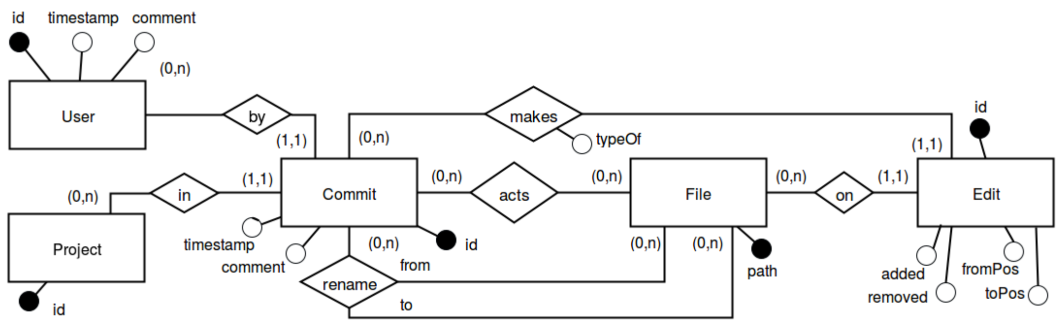
Deliverable 3.3 version 1 of the SHAPE project[1] reports work performed under Task 3.3 *Combined method for mining and extracting processes, related events and compliance rules from unstructured data*. This document meets milestone **(M3)** *Combined method for mining and extracting processes, related events and compliance rules from unstructured data*.

In previous deliverable D3.2 Cabanillas et al. [2015a], we studied the applicability of possible techniques for mining projects. This deliverable builds upon the work presented in the previous one in the following contents.

- We have developed a data model to access and analyze projects. We have identified the main entities and their relationships that are typical in version control systems (VCS) and have constructed a schema. This allows for storing data into a database and enables interesting queries. This approach is more scalable since it allows for more flexibility in getting insights into projects without the burden of encoding new approaches from scratch.
- We have studied possible text mining techniques in order to enrich the information provided by the mining algorithm in Cabanillas et al. [2015c]. Text mining can help to assign labels to the activities of a Gantt chart. In this deliverable we have tested a) topic modeling from emails; and b) semantic analysis of VCS comments.
- We have developed and tested an algorithm that combines both statistical information and commit-comments from VCS repositories to classify users according to roles.
- We are currently implementing a mining technique in Camunda that is aware of resource scheduling.
- We are developing a query language to access Camunda history logs through an SQL-like console.

This document is organized as follows. Section 2 presents the model we use to capture VCS data. Section 3 introduces shows how we intend to use text mining to support for better insights on projects. Section 4 presents a combined techniques for mining roles from VCS comments. Section 5 discusses an architecture and a prototype development that implements features from different work packages of SHAPE. Section 6 concludes the deliverable.

[1] https://ai.wu.ac.at/shape-project/



■ **Figure 1** Entity-Relationship Diagram from a VCS

2 Mining VCS for project-oriented processes and resources

In this section we analyze how we can mine useful insights from processes that use VCS. Engineering projects like the ones in SHAPE make use of version control systems to keep track of their documents. While the project evolves, the amount of information in these systems increases. New needs may emerge during the project lifetime or post completion. For instance, new rules and regulations may come into play, which have to be obeyed immediately; or a posteriori checks may be required from an auditor to manually check for unsafe patterns in the work that was done. Storing project data into a database allows to systematically access those data. Furthermore, it gives the flexibility to respond to new requirements without necessarily developing new algorithmic approaches to mine the data. For instance, new requirements can be checked by issuing a proper query to the database. In this section we will discuss our data model and show how we can use queries to analyze projects.

2.1 Data model

Here we give an overview of our data model. VCS are captured through the data model in Figure 1 using the Entity-Relationship notation from Chen [1976]. The entities and their relationships are described as follows.

Commit stores information about a checkout in the repository. Each checkout has an id or revision number, a timestamp and may have a comment. Comments are used from project members to describe their contribution. A commit can be of different types, e.g. it can store a merge between two commits or it can store file changes. Files, users, and edits on files are considered as separate entities in our model.

File represents a file that is present in the repository. A file is typically changed by a commit. We take into account also renames that are made to the file as a triple relation among two files and one commit.

Edit stores a modification of a *part* of the file. We use this entity to record where and how much a file was changed. This allows for fine grained analysis as we will see later in Section 2.2.

User is the resource who commits on the repository. A user has an id, a name and an email. Users of a project may issue an arbitrary number of commits.

Project is used to keep track of the project to which the set of commits belongs.

2.2 Querying data

With the data modeled as in Figure 1 it is possible to plan for several types of queries. We have used this model to store data into a MySQL[2] database. Database users are hence enable to issue suitable queries for their requirements. As an example, we show three possible queries. To this end we fed into our model the Camunda BPM[3] repository. We make the following queries.

1. Which are the contributions to the project of a particular user X? (Listing 1)
2. What changes have been made to a particular file over time? (Listing 2)
3. Who are the users that have worked on feature development? (Listing 3)

The queries have been written in SQL as follows.

```
SELECT name, Commit.id as CommitId, timeStamp, comment,
   linesAdded, linesRemoved, linesAdded-linesRemoved
FROM User, Commit, Edit
WHERE User.id = Commit.user_id
AND User.id = 1
AND Edit.commit_id = Commit.id
GROUP BY Commit.id
ORDER BY timeStamp ASC
```

Listing 1 Timeline of a user's contributions

```
SELECT sum(`linesAdded`),sum(`linesRemoved`), sum(
   linesAdded)-sum(linesRemoved) as delta,`timeStamp`,`
   comment`,`Commit`.`id`,`file_path`
```

[2] https://www.mysql.com/
[3] https://github.com/camunda/camunda-bpm-platform

```
FROM ‘Edit‘,‘Commit‘
WHERE ‘Edit‘.‘commit_id‘=‘Commit‘.‘id‘
AND ‘file_path‘="engine/pom.xml"
group by timeStamp
order by ‘timeStamp‘ ASC
```

Listing 2 Changes to the camunda engine configuration file

```
SELECT name, Commit.id as CommitId, timeStamp, comment,
   sum(linesAdded), sum(linesRemoved), sum(linesAdded-
   linesRemoved) as Delta
FROM User, Commit, Edit
WHERE User.id = Commit.user_id AND Edit.commit_id =
   Commit.id AND Commit.comment LIKE '%fix%'
GROUP BY Commit.id
order by timeStamp ASC
```

Listing 3 Users who worked on feature development

Listing 1 returns all the changes made to the repository from the user who has `id = 1`. Such user might have made multiple edits in one single commit. Thus, we aggregate by commit the numbers of lines added, lines removed and their difference which is also referred to as *delta*. Lastly we order these edit event from the oldest to the most recent.

Listing 2 shows the changes to a particular file. We chose the file named `"engine/pom.xml"` which is a configuration file that is supposed to change frequently due to updates, new releases (alpha, beta, etc) or tests. Here we want to see how the file changes over time. This can help to identify patterns that might exist in the development. For instance, we expect that when certain task is approaching completion, then the file changes become stable. This also why we order the results chronologically.

Listing 3 returns a list of user who have been working on a specific task. Here we exploit the commit-comments of the Camunda history log. Camunda comments include special tags like `fix(engine)`, `fix(platform)`, etc, for indicating what the changes were about. In this query we select all commit where the users have mentioned the word *fix* somewhere in the comment. Furthermore, we select the amount of changes and order the users chronologically.

2.2.1 Using timelines for project analysis

The queries of previous Section 2.2 can be exploited further to obtain more information. The extracted results can be exported and fed into other tools or data mining algorithms can be applied on top of them. As a showcase, we have analyzed our data with R[4] and have plotted the results for different queries.

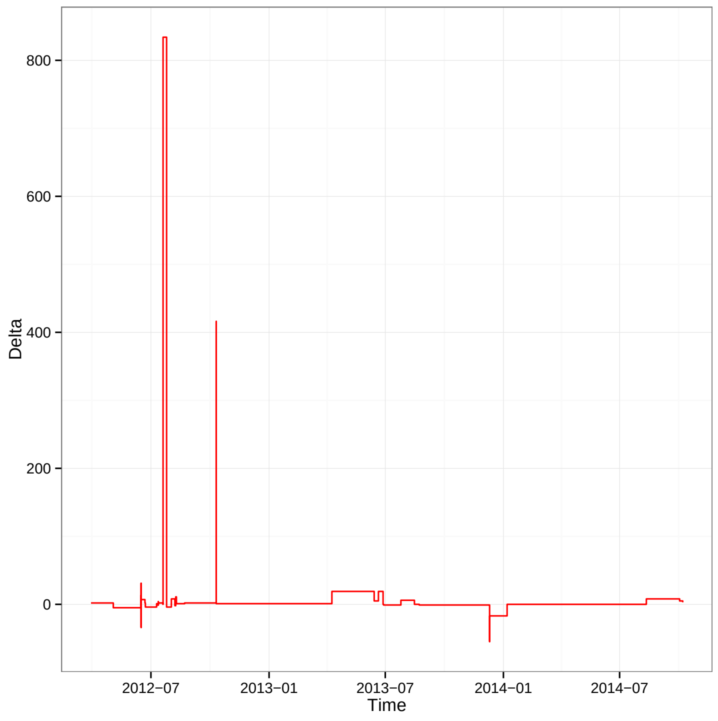
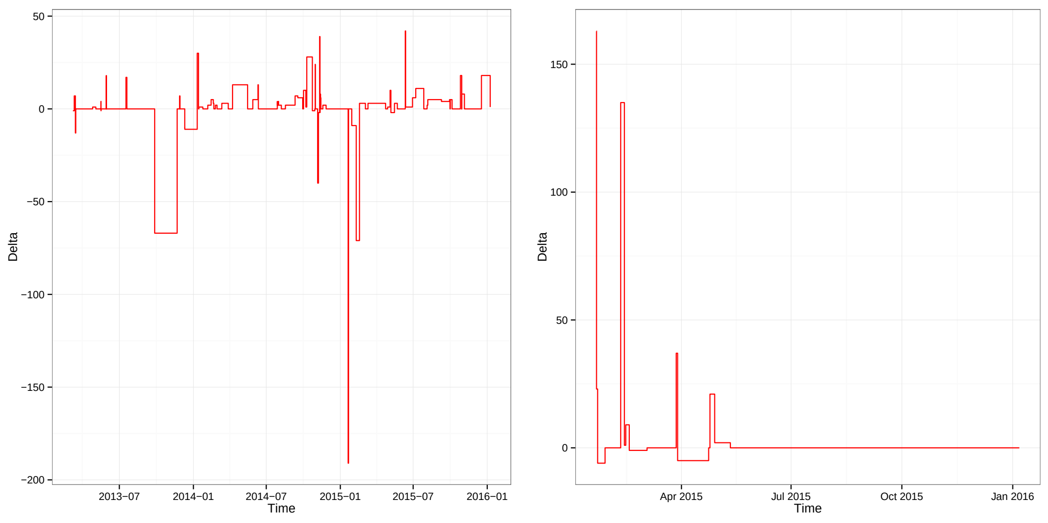


Figure 2 Profile of the changes made by one user. In the Y-Axis is reported the total amount of change that has been done by the selected user

Figure 2 shows the changes of a specific user over time on the repository. The Y-Axis is the amount of change *delta*, which is the difference between lines added and the lines removed from all files changed by the selected user. The data comes from the results of the query in Listing 1. It is now easy to see that user with `id = 1` has been generally not very active with two high peeks around his/her first appearance.

Figure 3 shows how we can use the query in Listing 2 to distinguish different types of work reflected by file changes. As an example, we run the query for two different types of files where the distinction is evident. Figure 3a reports the change over time of a file that belongs to the core development of Camunda.

[4] https://www.r-project.org/



(a) File under development **(b)** Database configuration file

Figure 3 An example of two different file-change patterns

Figure 3b plots changes of a database configuration file of Camunda. The graphs make it clear to see such change patterns.

We can use this kind of data to classify the type of work, the users and also gather further insights about the extent to which rules and regulations may be followed. Irregular patterns or artifacts that should be created, such as for instance validation reports, or comments about extensive testing that are not reflected by the changes in the VCS logs may be indicators of deviations. In Section 2.3 we give an example of rules and guidelines in software development projects that use VCSs.

2.3 Rules and guidelines

Rules and regulations have to be taken into account when it comes to check for process compliance. European Standards such as the EN 50128 software development process mus be followed in order ensure safety and reliability. Developer teams typically rely on VCS to manage their workflow. Thus, these systems can help to discover possible deviation from the desired flow. In the following, we will describe practical ways of using VCS for development. For details we point to online resources such as Driessen [2010] and Atlassian [2016].

2.3.1 Typical Workflows in VCS

Let us see an overview of four typical workflows on Git and discuss how they can be a) identified in a repository b) used to support mining.

2.3.1.1 Centralized Workflow

The centralized workflow uses a central point-of-entry for all the changes to the project. It works as follows. Each user has a local copy of the repository. All their edits to files get stored locally when they commit. To make a change visible in the main project streamline, users have to publish their changes (e.g. through commands `git push` or `svn commit`). Conflicts can occur when users try to push content that is already present in the main repository under another version. In such a case, users need to first `pull` the possible changes from the main trunk and resolve possible conflicts. Afterwards users can upload their edits. This workflow maintains a linear history.

2.3.1.2 Feature Branch Workflow

The feature branch workflow is based on the idea that the main repository, also called `master` branch, should contain only clean artifacts. In case of software development, it means that the features are developed in a dedicated branch and later merged into the main codebase. When a feature is ready, it is possible to issue a pull request. This triggers user review and conversation on the possible improvements of the feature before it gets accepted. The feature branch workflow is very flexible, allowing users to create as many branches as they want. However, sometimes this has the drawback that a common way of work is difficult to identify. The Gitflow Workflow discussed below provides a common pattern for managing feature development, release preparation, and maintenance.

2.3.1.3 Gitflow Workflow

The Gitflow Workflow defines a strict branching model designed around the project release. It assigns very specific roles to different branches and defines how and when they should interact. It uses individual branches for preparing, maintaining, and recording releases. Like in other models, Gitflow still uses a main `master` branch for developers to publish their local changes. The difference lies in the branch structure.

Figure 4 shows how dedicated branches are created for a specific use. The main branch `master` tracks the official software releases. The most important

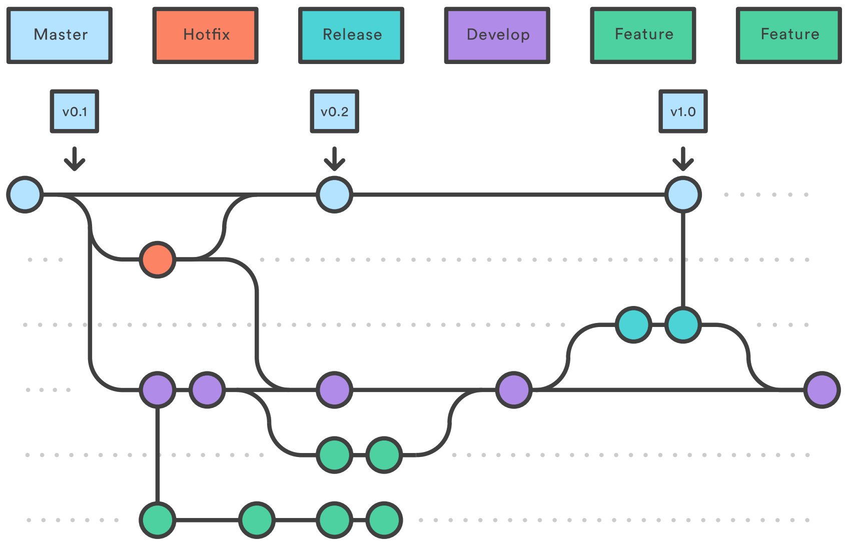


Figure 4 The Gitflow branches. Each branch serves to a specific purpose. Picture from Atlassian [2016]

branch after master is `develop`. `develop` serves as an integration branch for new features. Each feature is developed in its own branch that forks off `develop`. When a number of features are merged again into `develop`, the users may want to prepare a new release. To this end, they fork off a `release` branch from `develop`. A pull request to `master` may then trigger code review and discussions among developers. If the new release is accepted, it can then be published on the `master` branch and tagged with a version number. In case the end user discovers a bug and a quick fix becomes necessary, then a `hotfix` branch is created directly from `master`. As soon as the issue is solved, a new version is committed both in `master` and `develop`. In this way urgent problems are solved without going through the whole release cycle.

2.3.1.4 Forking workflow

In the workflows introduced above, users had a local copy of a central repository, namely a clone. Here, instead of a single server-side repository, each user has its own server-side copy of the official repository, namely a fork. Users have also a local copy derived from cloning their own server-side copies. The advantage here is that each user is concerned only with publishing on its own repository. A project maintainer can then pull from different projects into the official one, without having to give write permissions to the users. It works as follows.

- The project maintainer initializes the official repository
- Developers fork the official repository
- Developers clone their forked repositories
- Developers work on their features
- Developers publish their features
- The project maintainer integrates their features
- Developers synchronize with the official repository

Forking workflow implements a totally distributed workflow where developers are not tied to one team but they can share code with any other developers and any change can be merged into a project at any time.

3 Text mining projects

This deliverable uses text mining to exploit unstructured data that may give evidence on compliance deviation or gather new insights into the engineering projects of SHAPE. This is the case of e-mails and commit messages from VCS logs. In this section we consider two different approaches to text mining *i)* topic modeling; and *ii)* semantic annotation. We have evaluated both approaches with our data in order to better understand the possible future improvements of these techniques that fit our goals.

3.1 Topic models

Topic models are a set of methods that aim to discover a common theme in document collections. These document collections may talk about several topics. Topics are a set of words which are distributed over a vocabulary. Topic modeling algorithms are able to find topics and group them together into clusters. A simple yet extensively used topic model is given by latent Dirichlet allocation (LDA), published in Blei et al. [2003].

3.1.1 Topic modeling on SHAPE emails

In this section we show an application of LDA to the email collection from SHAPE. We have downloaded all the emails from the SHAPE mailing list and applied the approach described in Feinerer et al. [2008]. To this end we have created a single text file for each email. The data have then been imported into R. The procedure involves classic text-mining preprocessing steps such as stop-words removal, tokenization, etc, which are already provided by the `tm` and `topicmodels` libraries.

Table 2 Four topics extracted from SHAPE emails

Topic 1	Topic 2	Topic 3	Topic 4
axel	cristina	shape	saimir
saimir	cristinacabanillas	axel	deliverable
shape	shape	polleres	bala
polleres	next	information	tudor
stefan	information	simon	shape
week	claudio	business	architecture
cristina	meeting	meeting	review
will	deliverable	date	please
giray	agenda	steyskal	alois
paper	site	missing	project

Table 2 shows the topics for the emails of SHAPE. We cluster our documents into `k = 4` topics of which the ten most popular words are reported. Topics represents words which are highly correlated. For instance, in this case, people who have more closely worked together have exchanged more emails with each other. Hence, they get clustered under the same topic.

An advantage of topic models is that no annotation on the text is required as a prior step. However, sometimes topics alone are not very informative. Moreover, choosing the right number of topics is not straightforward. This work-package plans to investigate further on combining topic models with the approach from Bala et al. [2015] that mines Gantt charts. Topics can be mined only on the emails that where exchanged during active times slots that our found by process mining VCS logs. Then, such topics may be used to label the activities on the Gantt chart.

3.2 Semantic model

Here we present another text mining approach to learn from unstructured data. While topic models are statistical methods that do not take into account the relations among words or the structure of a sentence, semantic models allow us to exploit words hierarchies or similarities. Text from software repositories or emails can be semantically annotated and analyzed using such semantic approach. One interesting task that allows us to learn more about comments that people write in VCSs is to categorize their commits. We take inspiration from Leopold et al. [2012], who classify process labels into categories, to classify commit-styles. In this section we investigate to what extent it is possible to understand the commit-styles by analyzing them as possible labels of business process activities. We adapt the label-styles to commit-styles as shown in Table 3.

Table 3 Commit styles

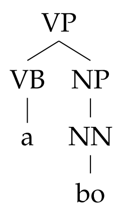
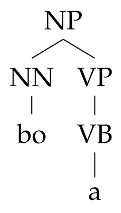
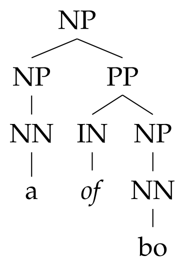
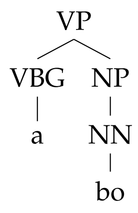
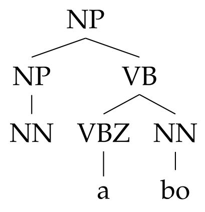
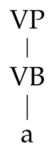
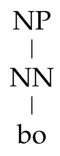
Commit style	Structure	Example
Verb-Object VO		update README.md
Action-Noun AN (np)		jar update
Action-Noun AN (of)		Creation of launchers
Action-Noun AN (ing)		reviewing code
Action-Noun AN (irregular)	`<irregular structure>`	(CAM-A14) bugfix#11
Descriptive DES		A left-over is removed now.
Only-Action OA		fix
Only-Object OO		feature

Table 4 Commit styles in Camunda and ProM

Property	**ProM**	**Camunda**
VO	28.00%	48.38%
AN	10.09%	4.66%
DES	.02%	.02%
OO	.91%	.64%
OA	.09%	.02%
N/A	60.89%	46.27%
Number of comments	19675	4208
Number of revisions	27574	5149
Avg No. of Sentences per Commit	0.71	0.82
Avg No. of Words per Comment	10.71	6.91
VCS	SVN	Git

Given the diversity of projects that can be retrieved from open source repositories and the fact that comments may have worse wording with respect to activity-labels, we expect the results to change from a repository to another.

We applied the rules in Table 3 to two different repositories.

- ProM - from academia
- Camunda - from industry

Table 4 shows the results of our classification of user comments. We can see that there are many comments that are not classified. This is due to their sentence structure. One big difference between activity labels VCS and comments is their length. As suggested by Table 4, in order to classify more comment-styles we need to take into account an average sentence size of at least ten words. However, the results still show how Camunda uses a better commit-style with respect to ProM, which gives more freedom to its users. In future work we plan to improve our classification method to be able to deal with the length of the commits from VCSs. It will be then possible to use commit-styles to classify users, work, or possible deviation (e.g. ambiguous description of work that may lead to process deviations).

4 Combined methods for mining user roles from VCS logs

This sections shows an approach to mine resource-roles combining information from both text and quantitative data from Version control system (VCS).

4.1 Background

Here we discuss the problem and related work.

4.1.1 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 plays a particular 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 to 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. There 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 about the resources who changed it and their comments, by looking into the VCS logs.

Figure 5 illustrates how people work in a project. They are assigned to defined tasks to which they contribute. Their contributions are part of generated artifacts. As time goes by, the number and the content of the artifacts on 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 goes along with 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 in 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.

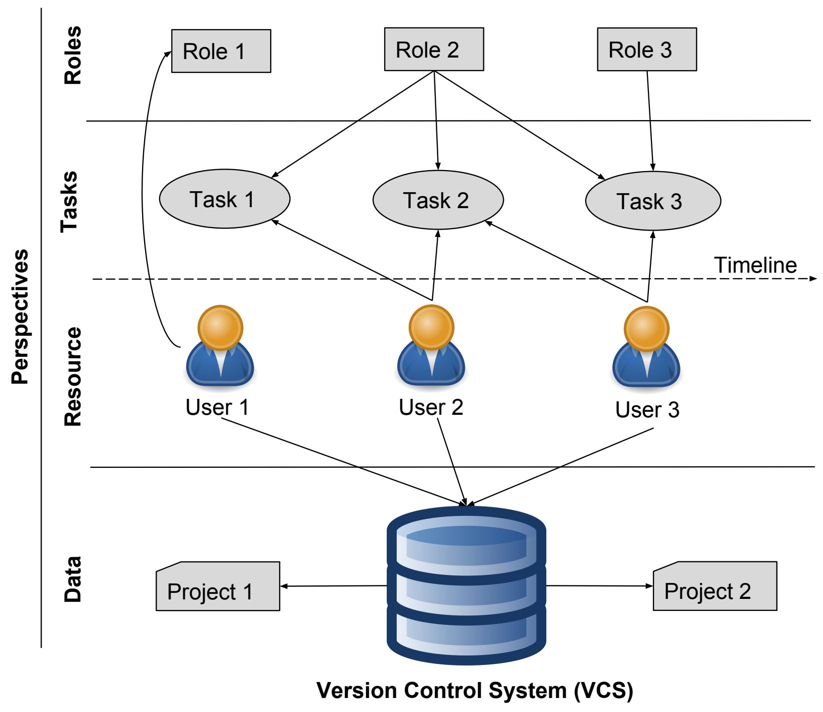


Figure 5 Software project and resources

Let us now see an example of how people use a VCS to collaborate in a project. Bob is a software engineer at Abc. ltd. He is working on the collaborative 'Project 1' with his colleagues. He adds a new module to 'demo' file, updates the file 'rule' and leaves respective comment. After performing the commit, the VCS log entry for this instance would look like the first log entry in the table with log ID 'abc123'. Alice is the UX designer working on the same project. Her job is to look after the user satisfaction from a design perspective. She makes changes in the 'setup' interface file and leaves the respective comment. After committing, the VCS log for this entry is represented in table with log ID 'jsh567'. Consequently, Bob had to make changes in the 'setup' programming module represented by the log ID 'aof082'. Some important insights into the VCS logs are: *a)* log ID is a unique id generated by VCS and changes each time a commit is performed by the user; *b)* user ID in VCS is universal and remains the same; *c)* comments, though useful for classification of users, are optional and committers might avoid it or write just a single word. We can clearly see here how VCS gives us granular information which can help us classify the users. However, it should be kept in mind that this is a very simple example of a VCS, and as the projects grow both in size and numbers VCS can get very complex.

Table 5 illustrates our running example. A unique log ID for each commit

Table 5 Example of a VCS log

Log ID	User ID	Repository	TimeStamp	Updated Files	Comment
abc123	bob14	Project 1	2014-10-12 13:29:09	Demo.jar rule.txt	Added new module to demo and updated rules
jsh567	alice01	Project 1	2014-11-01 18:16:52	Setup.exe	Modified the setup interface
547	kriss	Project 2	2015-06-14 09:13:14	Todo.doc	Update the application interface
anfn876	s_tony	Project 3	2015-07-12 15:05:43	graph.svg todo.doc	Define initial process diagram & listed remaining tasks
aof082	bob14	Project 1	2014-11-05 10:12:47	setup.exe	Updated

performed by the user. Furthermore we can see the information on which repository the commit was performed, at what time and date, and which files were subsequently updated. User comments are dependant on many variables like personality, requirements etc and therefore sometimes they can be blank or contain all sorts of irrelevant information user might mention.

4.1.2 Related Work

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

4.1.2.1 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.

Mining software repositories approaches. In the area of MSR, Yu and Ramaswamy [2007] 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. [2008] 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. [2008] classify developers contribution based on lines of code (LOC) changes and map infer activities from them. Begel et al. [2010] developed the Codeboook software tool a utility for fining experts. They use a social network approach that combines sources from people, artifacts, and textual allusions to other people. Ying and Robillard [2014] 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. [2014] investigate user roles in innovation-contest communities. They use quantitative methods to analyze user activity logs and interpretative to categorize qualitative comments into classes.

Process mining approaches. Efforts have been done to analyze software repositories with process mining techniques. Rubin et al. [2007] 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 perspective, 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. [2011] developed FRASR, a framework for analyzing software repositories. FRASR can be used in order to transform VCS logs data into the XES (Verbeek et al. [2010]) data format that can be further analyzed with process mining tools like ProM[5]. Song and van der Aalst [2008] focus on three types of organizational mining *i)* organizational model mining, *ii)* social network analysis, and *iii)* information flows between organizational entities. Schönig et al. [2015a] propose a mining technique to discover resource-aware declarative processes.

4.1.2.2 Unstructured data

Maalej and Happel [2010] 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. [2012] developed an identity merging algorithm based on Latent Semantic Analysis (LSA) to disambiguate user emails. Licorish and MacDonell [2014] mined developer comments to understand their attitudes.

4.1.2.3 Other related work

The term *role mining* often points to role mining algorithms based on role-based-access-control (RBAC) systems. These algorithms takes as input predefined roles that are given as a matrix, where each users are assigned to access permissions. A number of algorithms have been developed to mine roles from RBAC systems alone (Lu et al. [2015], Frank et al. [2013]) or combining their data with process history logs (Baumgrass et al. [2012]). A survey of existing techniques and algorithms can be found in Mitra et al. [2016]. Our work is disjoint from this class of algorithms as VCS does not contain access control information. Bhattacharya et al. [2014] propose a contributor graph-based model. By constructing both 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. [2013] use a grounded theory (GT) approach to study agile teams.

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

Their work unfolds the roles of *mentor*, *coordinator*, *translator*, *champion*, *promoter*, and *terminator*.

This work builds upon existing literature in that it gather insights on organizational level like in Rubin et al. [2007] and Song and van der Aalst [2008] 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. attribute roles to resources. Lastly, this approach differs from Bhattacharya et al. [2014] and Hoda et al. [2013] since we further adopt NLP techniques.

4.1.3 Research Questions

As mentioned in the previous section, the focus of this work lies on mining and analyzing properties of the users of VCS. The users belong to certain classes and these classes have different commit message styles. Based on this assumption, the following research questions are defined:

RQ1 *What classes can be assigned to users of VCS?* The first research question scrutinizes if a classification of the users of a VCS is possible only based on the information provided by the log files. The classification separates the users into clusters. To this extent the most expressive features and combinations of these features have to be identified. These clusters are then further analyzed by means of the next research question.

RQ2 *How can we map users types to classes?* Based on the created clusters meaningful classes are derived. The chosen features influence the types of classes that can be created and there has to be an intelligible distinction between those classes. This research question also answers how detailed this distinction can become and it creates behavioral profiles for the class members, based on the analyzed features.

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

4.2 Approach

Commit messages contain metadata and content. For analysis of users and roles both parts are important. Metadata includes the author, which is essential for gaining a unique identifier of the committer. The content gives some indications on roles. Even if data is fetched already in the right format, it is still not ready to be analyzed. Information cleaning is as important as the extraction. Changed roles, one time committers, or several identities for the same use, are some of the challenges that need to be handled.

Once data is prepared, the analysis can start with a direct approach of connecting one commitment message to one user and his/her corresponding role. While this 1:1 relationship is not always available, there are other possible analyses like the commit based approach. It is an indirect approach because in it a list of roles to every commit message. By linking the lists of roles to the authors the role of the user is predicted.

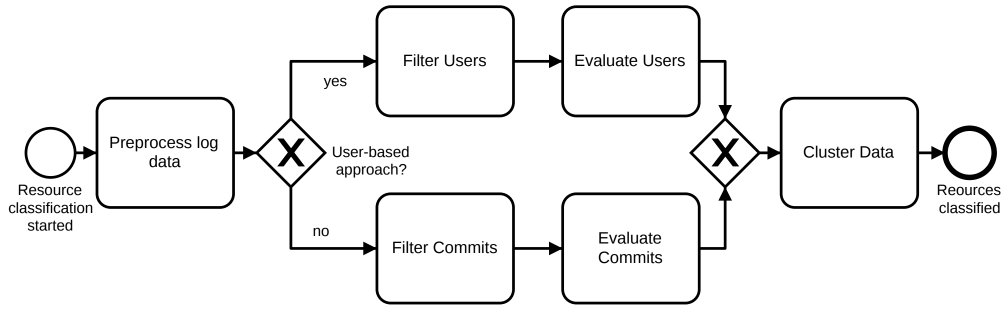


Figure 6 Approach to role classification from VCS logs

Figure 6 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 user or to commits. Both the 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 4.3 describes both the approaches in detail.

4.3 Implementation and evaluation

One solution for the outlined problem is an algorithmic approach in form of building a script using Python. This script automatically fetches, processes and analyzes 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[6].

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

Machine Learning. The Scikit Learn module is utilized for machine learning. It is built on matplotlib, numpy and scipy. It offers algorithms and graphical representations for classification, regression, clustering, dimensionality reduction, model selection and preprocessing[7]. We used decision treess (DTs) for classification and regression. DTs are supervised learning method whose goal is to create a model that predicts of a target feature of variable by inferring existing rules from the data.

Natural Language Processing. For natural language processing the Natural Language Toolkit (NTLK) offers methods to extract additional information out of everyday communication. The "Bag of words" technique analyzes word occurrences, but also more sophisticated methods, which develop a human understanding of communicated messages, can be utilized [8].

After choosing the tools, a very important step required prior to the actual implementation is the selection of data. While the number of accessible code repositories available via repository hosting services is quite huge, picking a dataset of appropriate size and quality proves rather challenging. There have to be some considerations to be made for choosing the right repositories:

- Number of repositories: Classification on a single repository might not produce a generally applicable result, whereas using more repositories increases complexity.
- Size of individual repositories
- Role information: This is important for certain steps in the classification task and for verification of results.
- Differences in organisational aspects of the projects: The type of the development team (professional or hobby) type of software (proprietary or open-source) and type of platform (private server or public repository hosting service) might have influences on the way repositories are used.
- Differences in version control systems

For this project three repositories are analyzed, each with multiple thousand commits:

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

[7] http://scikit-learn.org/stable/
[8] http://www.nltk.org/book/ch00.html

3. Camunda GitHub project repository. Open source software. VCS: Git

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

First of all, browsing through the commit messages created an understanding of the structure of the log files, the type of information available for extraction and the way VCS commits are typically used in the different projects. They contain the most useful information but are also the least predictable factor, because there are no general rules on how to formulate such a message. 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.

The first step of creating the algorithm, once the the log files were obtained and their basic structure was known, is to transform them into a common in-memory object model. Three different functions, providing the same output for different input formats, were required to cover all three systems (Git, Svn, Mercurial). This model consists of one 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 it is possible to derive an additional model consisting of the users, by aggregating the information for each author name occurring in the commits.

For further analysis and classification steps, efforts have been split up into two approaches. The first one focuses purely on the users and the features deductible for them. We soon realized though that this method, while providing some promising insights, had certain limitations and left out some potentially valuable information. Therefore a second approach was incorporated taking a closer look at the individual commits. The two approaches are outlined in more detail in the following sections.

4.3.1 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 finding and calculating features for the users where useful differences between classes might be occurring. An explorative approach is required to find the meaningful ones. The calculated and analyzed features are:

- 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)
- Commits message length: Average length of commit messages in number of words or characters
- Occurrence numbers of certain keywords: How often a certain word (e.g. "test", "fixed", etc) is used by a user, relative to the total number of commits.
- Number of added/modified/deleted files
- Occurrence number of file formats: How often a file with a certain format (e.g. .java, .py, .html...) are modified by a user, relative to the total number of modified files.

In order to find useful clusters, an experimentation with different combinations of features is necessary. For each of these combinations the optimal cluster number has to be found. While the clustering and optimizing tasks can be done programmatically, the clusters have to be evaluated manually. This makes it impossible to test all possible combinations of features, especially when including the keyword occurrence numbers, as the list of keywords with potential value was far too large. Combinations that promise useful results are selected. Evaluation is done using plots as well as looking at the raw data to find connections between the identified clusters and the existing classes in the sample data.

From the clusters generated with this method classification models are built using the decision tree classification method. The decision tree 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.

4.3.2 Commit-based Approach

There are multiple reasons leading to the decision of adopting 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 executes many tasks not belonging to his 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 lead to the idea of classifying individual commits.

The algorithm iterates over commits and tries to assign types to them. The types are assigned based both on the analysis of the commit message, and the file extensions. The 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 6.

Table 6 Keywords used to classify the type of work

Test	**Development**	**Web**	**Tool**	**Maintenance**	**Refactor**	**Documentation**	**Design**
test	implement	web	tool	bugfix	refactor	documentation	style
tested	implemented	spring	library	bugfixes	refact	documented	styles
testing	implementation	http	libraries	fix	refactored	javadoc	styling
tests	implementing	https	framework	fixed	refactoring’	readme	icon
testcase	implements	rest	dependency	fixes	refactorings	userguide	icons
testcases	improved	html	dependencies	fixing	rename	user	font
test	improving	css	upgrade	patch	renamed	guide	layout
case	update	servlet	upgrade to	patched	renaming	tutorial	layouts
test cases	updated			cleanup	move	faq	layouted
cases	updating			cleanups	moved	translation	layouting
unittest	script			clean up	revert	translate	file types:
unit	scripting			clean	reverted	translated	png
test				cleaned	reverting	translating	svg
integrationtest				cleaning		doc	jpg
integration						i18n’	
test						file types	
fitnesse						txt, docx, text, tex, pdf	

Build	**Data**	**Backend**	**Addition**	**Removal**	**vcsManagement**	**Automated**	**Merge**
build	data	engine	added	delete	svn	(starts with)	(starts with)
building	database		add	deleted	git	Automated Release	Merge
compile	sql		adding	deleting	mercurial	Automated Nightly	merge
compiled	postgres		new	remove			#merge’
compiling	postgre		create	removed			
release	postgresql		created	removing			

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.

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. 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 oficial roles but contribute in the VCS mainly in form of minor, supportive tasks.

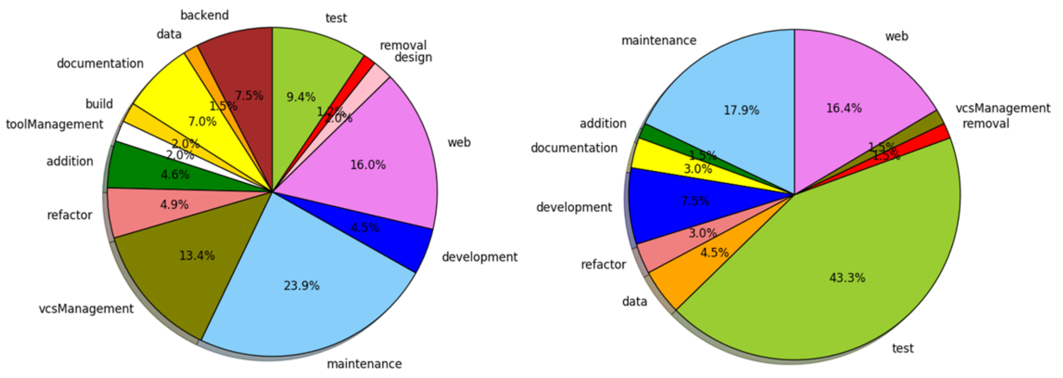
The classification task was done in two ways: 1. Manual analysis of the table and derivation of rules by looking for similarities between users with the same role. 2. Automated classification in line with our original concept. For the latter the decision tree method is used. In this case the commit type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

4.3.3 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 other developers group pushes every small



(a) User profile for a backend developer **(b)** User profile for a tester

Figure 7 Commit distributions of the Infinica dataset

change into the repository separately 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" is already enough 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 quite 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 7a and Figure 7b, 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 8. The developers (red and dark blue) 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 red cluster is comprised of developers which push every change to the repository.
- **Developers - heavy commits:** The dark blue cluster contains developers who work on their local machine and push less frequently. However, they have

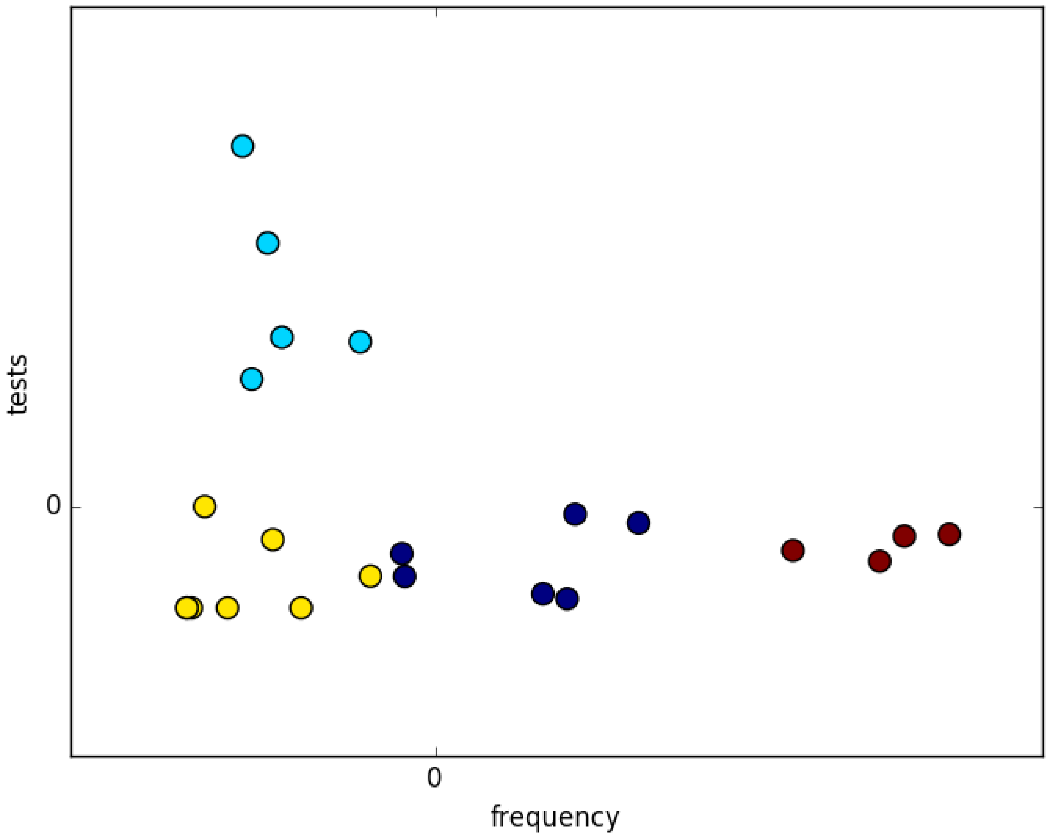


Figure 8 Scatterplot with colorcoded clusters (Infinica data set)

bigger commits containing more changes.

- **Testers - technical:** Testers focused on automating the testing process are in the light blue cluster. They are solely senior developers but not all of them are situated in this group.
- **Testers - non technical:** The yellow cluster is comprised of less technical testers which tend to do more manual testing. Additionally, it includes the special service team.

Table 7 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 left out here due to a lack of meaningfulness. 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

Table 7 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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

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 9.

- **Core developers:** The employed users of the ProM project are situated in the light blue cluster to the right. Their commit frequency and absolute number 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 dark blue 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 red cluster at the bottom left.
- **Testers:** Despite the lack of testers in the ProM data set two have been identified as such.

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 users (14%) were regarded as testers, which fits to our results in the user-based approach. One (2%) remaining user was classified as a web developer. As the ProM project has no web component,

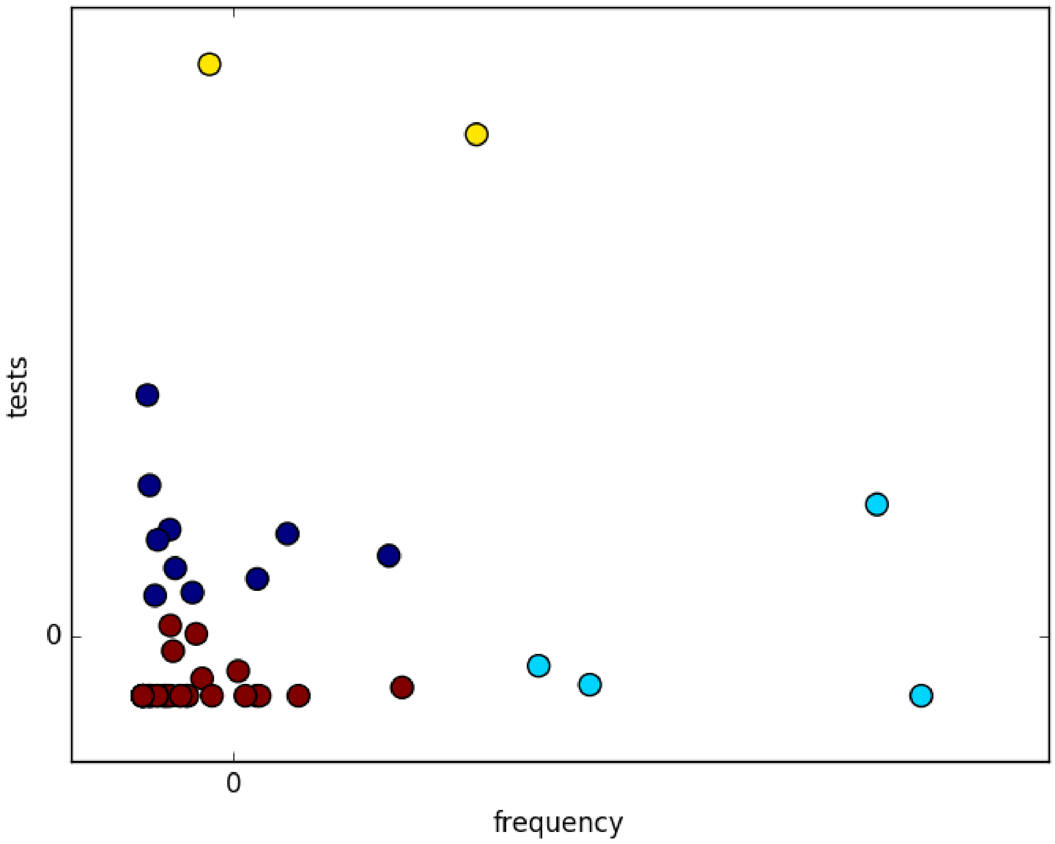


Figure 9 Scatterplot with color-coded clusters, ProM data set

it makes sense that there are no web developers identified. The one we found has only 5 commits, 2 of them web commits so this (potential) misclassification is ignored.

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 just 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 10.

- **Core developers:** The red 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 light blue cluster. They are refining their own committed code or they are extending the product in different areas.

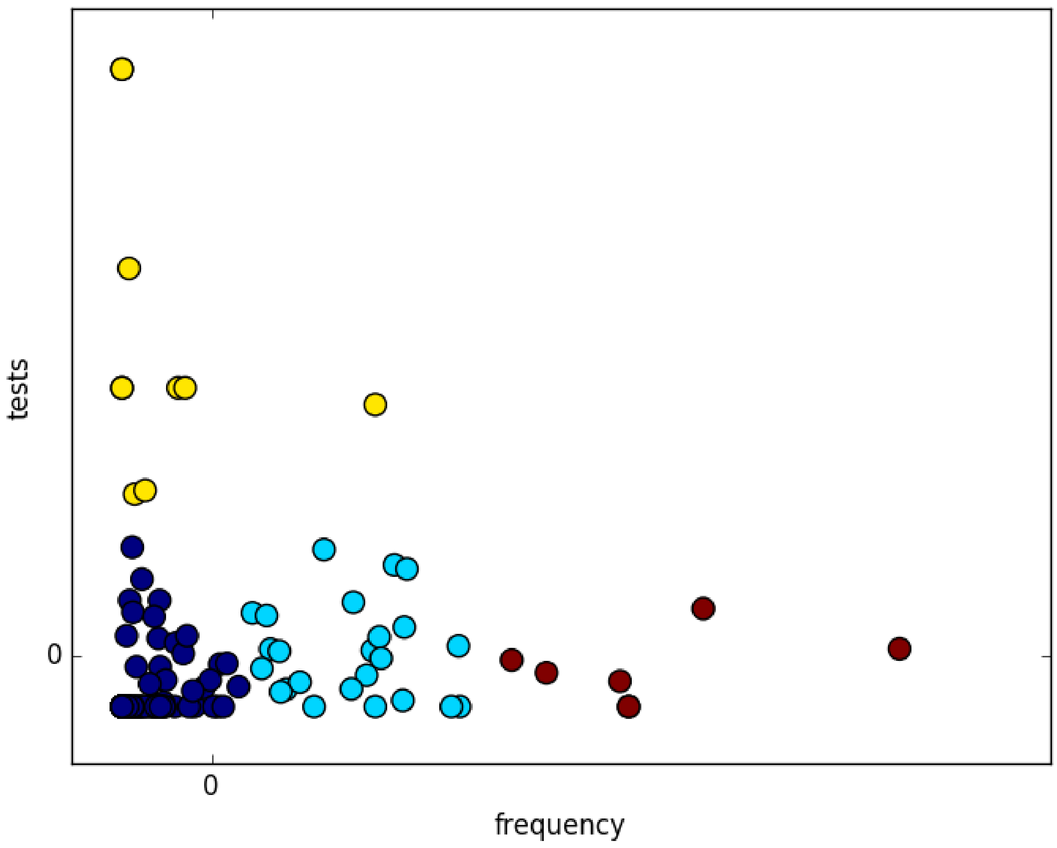


Figure 10 Scatterplot with colorcoded clusters, Camunda data set

- **One-time developers:** Similar to the ProM data set the majority of the users belongs to this group and it is represented by the dark blue cluster. Code usefull or not is committed typically once and there is no lasting commitment.
- **Testers:** Also testers have been identified in the yellow cluster. They already have a lower frequency than developers and therefore it is difficult to make an estimation about their commitment.

For the Camunda data the case is similar. Out of 66 total users, 49 (74%) were regarded as developers, the high number being again explained by the type of project and confirmed by the contact person we spoke to. Ten (15%) testers were found, mostly in line to our knowledge 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.

For the user-based apparoach 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

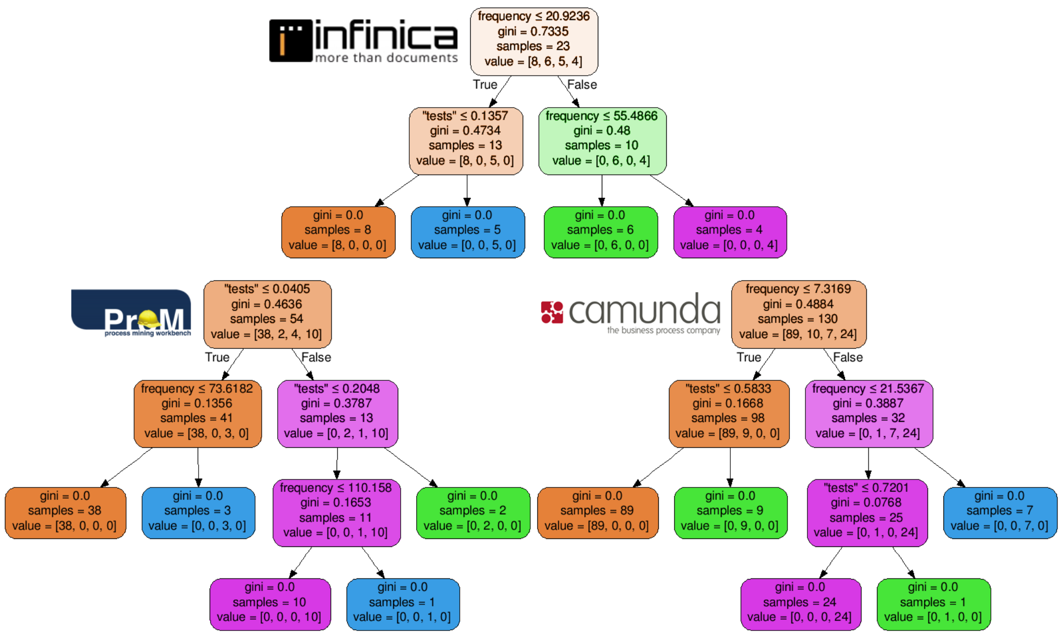

infinica more than documents
frequency ≤ 20.9236 gini = 0.7335 samples = 23 value = [8, 6, 5, 4]
True
False
"tests" ≤ 0.1357 gini = 0.4734 samples = 13 value = [8, 0, 5, 0]
frequency ≤ 55.4866 gini = 0.48 samples = 10 value = [0, 6, 0, 4]
gini = 0.0 samples = 8 value = [8, 0, 0, 0]
gini = 0.0 samples = 5 value = [0, 0, 5, 0]
gini = 0.0 samples = 6 value = [0, 6, 0, 0]
gini = 0.0 samples = 4 value = [0, 0, 0, 4]
ProM process mining workbench
"tests" ≤ 0.0405 gini = 0.4636 samples = 54 value = [38, 2, 4, 10]
True
False
frequency ≤ 73.6182 gini = 0.1356 samples = 41 value = [38, 0, 3, 0]
"tests" ≤ 0.2048 gini = 0.3787 samples = 13 value = [0, 2, 1, 10]
gini = 0.0 samples = 38 value = [38, 0, 0, 0]
gini = 0.0 samples = 3 value = [0, 0, 3, 0]
frequency ≤ 110.158 gini = 0.1653 samples = 11 value = [0, 0, 1, 10]
gini = 0.0 samples = 2 value = [0, 2, 0, 0]
gini = 0.0 samples = 10 value = [0, 0, 0, 10]
gini = 0.0 samples = 1 value = [0, 0, 1, 0]
camunda the business process company
frequency ≤ 7.3169 gini = 0.4884 samples = 130 value = [89, 10, 7, 24]
True
False
"tests" ≤ 0.5833 gini = 0.1668 samples = 98 value = [89, 9, 0, 0]
frequency ≤ 21.5367 gini = 0.3887 samples = 32 value = [0, 1, 7, 24]
gini = 0.0 samples = 89 value = [89, 0, 0, 0]
gini = 0.0 samples = 9 value = [0, 9, 0, 0]
"tests" ≤ 0.7201 gini = 0.0768 samples = 25 value = [0, 1, 0, 24]
gini = 0.0 samples = 7 value = [0, 0, 7, 0]
gini = 0.0 samples = 24 value = [0, 0, 0, 24]
gini = 0.0 samples = 1 value = [0, 1, 0, 0]

Figure 11 Decision trees, all three data sets

previous clusters. The associated increase or decrease in commit frequency can not be linked to the development.

For the commit-based approach 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 commi type, for which we already expressed our doubts.

The corresponding decision trees for the previous presented k-means clustering shown in Figure 11 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. In the second step these 2 classes are then divided into subclasses. On the contrary, the other two trees start separating the subsets right away in different orders.

The decision tree model is depicted in Figure 12. 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

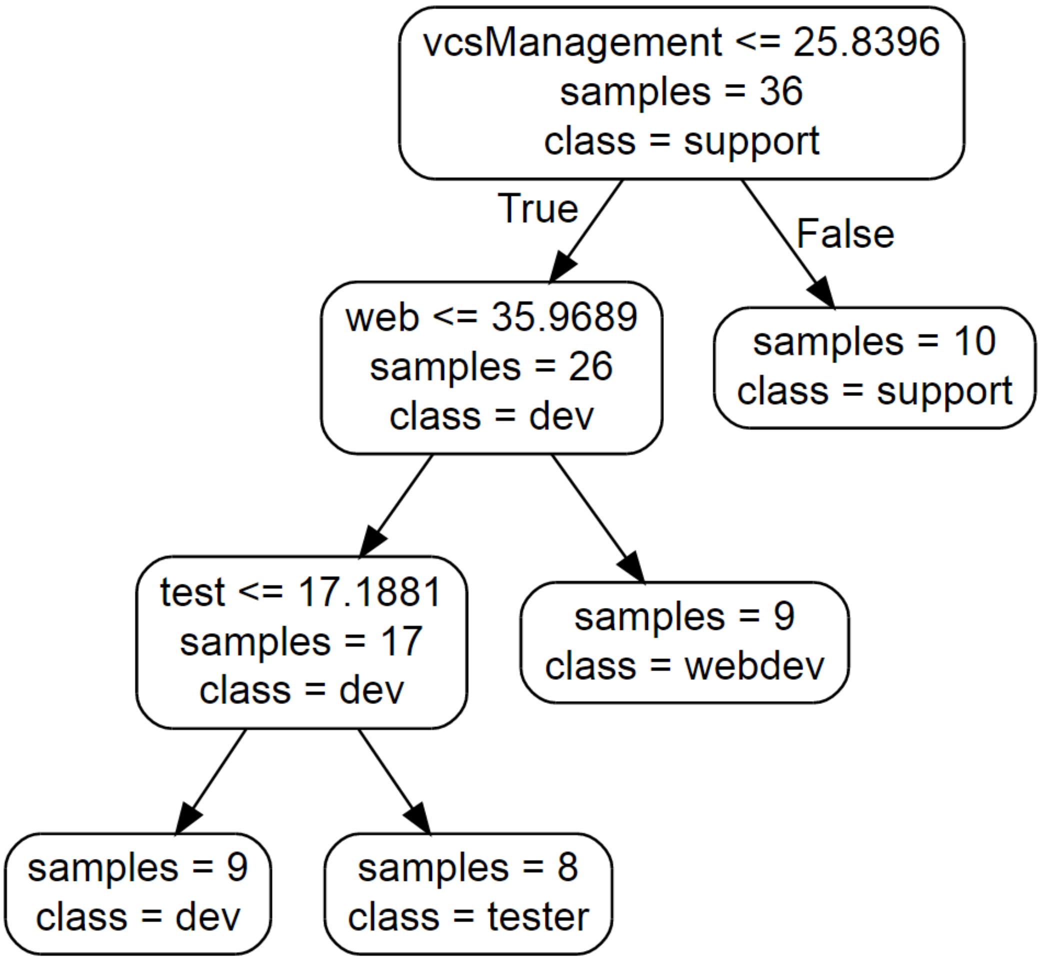


Figure 12 Decision tree, commit based model

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 just 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 of course that it can be applied to other data sets very efficiently. Doing that with the Camunda and ProM data sets provided some verification for our decision tree model. We

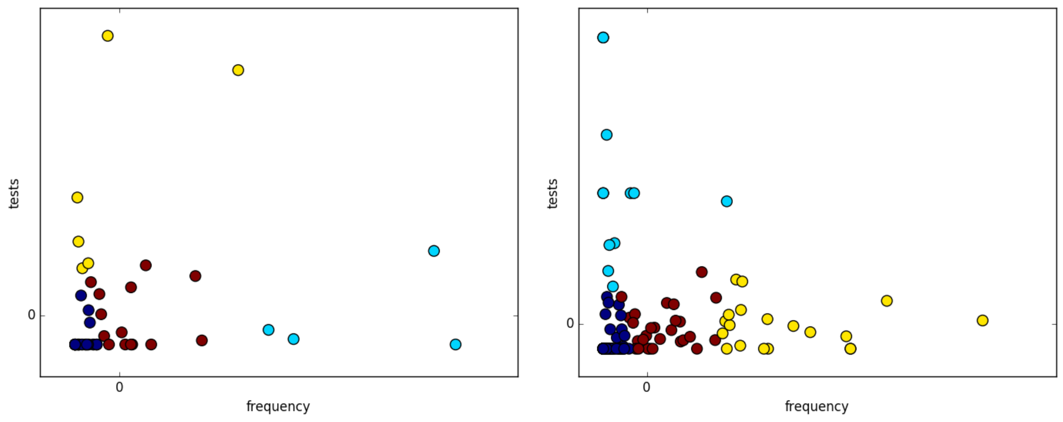


Figure 13 Classication based on Infinica training set: Prom, Camunda

only included users with 5 or more commits into the cross validation.

When validating the model with the Infinica data set and then predicting the ProM and Campunda data set only moderate results can be achieved with the user-based approach as shown in Figure 13. 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 13 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 13 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.

4.3.4 Discussion

Throughout this project we discovered a number of research directions to go 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 can be answered with: Yes, it is possible to classify users based on the information usually found in VCS logs. Our results prove 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 because the ways commit messages are used 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.

4.3.5 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.

Acquiring the right data proved to be more challenging than expected. While there is a large selection of open source projects, acquiring useful information for potential classes can be difficult. Furthermore we have to assume that there are significant differences between open source and other repositories, which need to be further investigated for making a clear statement of the general representativeness of our data and our results.

A definite limitation for the whole topic of analyzing VCS logs is the fact that commit messages are used differently between users, repositories and teams. We have to assume that it is practically infeasible to find one classification model, which produces valid results for any VCS repository, because there can be contradictions in what certain statements mean. Even worse, some users even leave the commit messages empty, which means no classification is possible for this case.

4.4 Future Work

The exposed approach is the preliminary stage on which we plan to build upon a more comprehensive solution of the problem of role discovery from VCS. There are many ways to extend our approaches as well as other potential methodologies for classification. On the technical level, different methods of clustering and classification from the machine learning domain could be used. More conceptual additions would be for example the use of additional features and feature combinations.

There is also much potential for further refining our proposed methods. For instance, we would use more extensively the natural language toolkit for processing the commit messages. So far we only used the more basic functionality of the language processing tools. In this research we focused mainly on the user roles for classification but there might be other classes, for example discriminated by the experience of users. Significant improvements would be possible by analyzing different categories of VCS repositories. Especially log files of large software companies, where many different roles are formally defined and executed would be interesting.

5 Implementation on Camunda

This section briefly describes two prototypes that are currently under development.

5.1 Combining process mining and resource allocation

In this prototype, we aim to bring together functionality from mining and resource allocation (Cabanillas et al. [2015b]). Figure 14 shows the architecture of the prototype. Three main components play a role here.

Reasoner. The reasoner's task is to schedule human and non human-resources. As soon a plan is ready, the process activities as well as other resources (e.g. laboratory) are mapped together into a schedule. The schedule is a set of assignments like in Listing 4.

```
assign(amy,t_submit_equipmental_rental_request,0,1,1,bp_buildit,i1)
assign(tom,t_select_suitable_equipment,1,2,2,bp_buildit,i1)
assign(tom,t_check_availability,2,4,3,bp_buildit,i1)
assign(drew,t_review_rental_request,4,5,4,bp_buildit,i1)
```

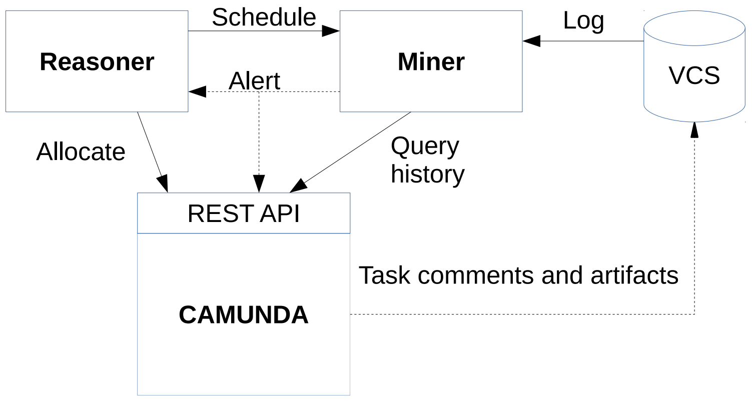


Figure 14 General architecture

```
assign(tom,t_create_po,5,7,5,bp_buildit,i1)
```

Listing 4 Example of a schedule

Camunda Camunda is the business process management system (BPMS) used by SHAPE. All the process instances that run into Camunda, and data from all the past executions of processes are stored into the Camunda logs. These logs can be queried through APIs. Listing 5 is an example of what is returned when querying the history using the Camunda REST APIs.

```
[{"id":"22f6751b-b526-11e5-a242-183da2307e3c","parentActivityInstanceId":
    "22f6751a-b526-11e5-a242-183da2307e3c","activityId":"sid-71A66E94-8
    B8F-4F91-A393-1B52839223A1","activityName":"Exa delivered","
    activityType":"startEvent","processDefinitionKey":"sid-68bc060f-be0e
    -43d0-9d23-689df97192c5","processDefinitionId":"90d5639b-b521-11e5-
    bf1a-183da2307e3c","processInstanceId":"22f6751a-b526-11e5-a242-183
    da2307e3c","executionId":"22f6751a-b526-11e5-a242-183da2307e3c","
    taskId":null,"calledProcessInstanceId":null,"calledCaseInstanceId":
    null,"assignee":null,"startTime":"2016-01-07T11:05:10","endTime":"
    2016-01-07T11:05:10","durationInMillis":1,"canceled":false,"
    completeScope":false},{"id":"EndEvent_1:0dcd7116-8688-11e5-bd23-183
    da2307e3c","parentActivityInstanceId":"d8219f97-8686-11e5-bd23-183
    da2307e3c","activityId":"EndEvent_1","activityName":"Exam handled","
    activityType":"noneEndEvent","processDefinitionKey":"exam","
    processDefinitionId":"exam:2:aeb31626-8686-11e5-bd23-183da2307e3c","
    processInstanceId":"d8219f97-8686-11e5-bd23-183da2307e3c","
    executionId":"d8219f97-8686-11e5-bd23-183da2307e3c","taskId":null,"
    calledProcessInstanceId":null,"calledCaseInstanceId":null,"assignee":
    null,"startTime":"2015-11-09T03:17:40","endTime":"2015-11-09T03:17:40
    ","durationInMillis":0,"canceled":false,"completeScope":true}
```

Listing 5 Example Camunda history data

Miner The miner is the component that implements the mining algorithm. This component is able to communicate with Camunda by using the REST APIs. At the same time is implemented as a service that listens to incoming schedules from the Reasoner. Given a schedule and data from both Camunda and VCS logs, the Miner can trigger alerts in case of deviation from the schedule.

We plan to use this prototype as a demo at the next SIMPDA[9] conference.

5.2 Querying process history

We are developing a tool that will allows for SQL-like queries on top of Camunda logs. The approach involves mapping Camunda's database schema to RXES (van Dongen and Shabani [2015]). In addition to this we are also developing a tool that can map from RXES to XES Verbeek et al. [2011] and we plan to use this tool with the approach from Schönig et al. [2015b] in order to make it fully compatible with the RXES standard.

6 Conclusions

In this deliverable we have studied and implemented several approaches to gather insights from project by combining data from structured and unstructured sources. We showed how we can capture project data through a schema. This approach is flexible: it allows to gather different insights by simply changing the query. We also showed that we can further elaborate the query results and get better understanding of the project. Text mining approaches have also been evaluated. Using semantic models for VCS seems promising. We plan to improve on this by working on a better categorization of the comments. Combined techniques that take into account additional data to the comments have been presented. We used this method to classify users according to roles. The results have been validated with experts that work on the projects under analysis. In order to integrate the approaches from different work packages, we are developing a tool that combines functionalities from Cabanillas et al. [2015a] and Cabanillas et al. [2015b] and use the Camunda BPMS platform. The resulting prototype will be presented in the next SIMPDA 2016 conference. Furthermore, we are developing two approaches that will allow to easily query the Camunda history log for complex insights. These methods are parts of two master theses.

[9] http://simpda2016.di.unimi.it/

References

Omar Alonso, Premkumar T. Devanbu, and Michael Gertz. Expertise identification and visualization from CVS. *Proc. 2008 Int. Work. Min. Softw. Repos. - MSR '08*, page 125, 2008. ISSN 02705257. 10.1145/1370750.1370780. URL http://portal.acm.org/citation.cfm?doid=1370750.1370780.

Atlassian. Comparing workflows, 2016. URL https://www.atlassian.com/git/tutorials/comparing-workflows.

Saimir Bala, Cristina Cabanillas, Jan Mendling, Andreas Rogge-Solti, and Axel Polleres. Mining project-oriented business processes. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management*, volume 9253 of *Lecture Notes in Computer Science*, pages 425–440. Springer International Publishing, 2015. ISBN 978-3-319-23062-7. 10.1007/978-3-319-23063-4_28.

Anne Baumgrass, Sigrid Schefer-Wenzl, and Mark Strembeck. Deriving process-related rbac models from process execution histories. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pages 421–426. IEEE, 2012.

Andrew Begel, Yit Phang Khoo, and Thomas Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *2010 ACM/IEEE 32nd Int. Conf. Softw. Eng.*, volume 1, pages 125–134, 2010. ISBN 978-1-60558-719-6. 10.1145/1806799.1806821.

Pamela Bhattacharya, Iulian Neamtiu, and Michalis Faloutsos. Determining Developers' Expertise and Role: A Graph Hierarchy-Based Approach. *2014 IEEE Int. Conf. Softw. Maint. Evol.*, pages 11–20, 2014. ISSN 1063-6773. 10.1109/ICSME.2014.23.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Cristina Cabanillas, Saimir Bala, Jan Mendling, and Axel Polleres. Mining processes, resource consumption and witnesses for task completion from logs. Deliverable D3.3 (M3), 2015a.

Cristina Cabanillas, Giray Havur, Jan Mendling, Axel Polleres, Vadim Savenkov, and Alois Haselboeck. Unified semantic model and reasoning techniques for mining and monitoring process-relevant data. Deliverable D3.3 (M3), 2015b.

Cristina Cabanillas, Jan Mendling, Axel Polleres, and Saimir Bala. Requirements for process, resource and compliance rules extraction from text. Technical Report 1, 2015c.

Peter Pin-Shan Chen. The entity-relationship model&mdash;toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, March 1976. ISSN 0362-5915. 10.1145/320434.320440. URL `http://doi.acm.org/10.1145/320434.320440`.

Vincent Driessen. A successful git branching model. *URL http://nvie. com/posts/a-successful-git-branching-model*, 2010.

Ingo Feinerer, Kurt Hornik, and David Meyer. Text Mining Infrastructure in R. *J. Stat. Softw.*, 25(5):1–54, 2008. ISSN 15487660. citeulike-article-id:2842334. URL `http://www.jstatsoft.org/v25/i05`.

Mario Frank, Joachim M Buhman, and David Basin. Role mining with probabilistic models. *ACM Trans. Inf. Syst. Secur.*, 15(4):15, 2013.

Johann Füller, Katja Hutter, Julia Hautz, and Kurt Matzler. User Roles and Contributions in Innovation-Contest Communities. *J. Manag. Inf. Syst.*, 31(1):273–308, 2014. ISSN 07421222. 10.2753/MIS0742-1222310111.

Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. Measuring developer contribution from software repository data. In *Proc. 2008 Int. Work. Conf. Min. Softw. Repos.*, pages 129–132. ACM, 2008.

Rashina Hoda, James Noble, and Simon Marshall. Self-organizing roles on agile software development teams. *Softw. Eng. IEEE Trans.*, 39(3):422–444, 2013.

Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark G J Van Den Brand. Who's who in Gnome: Using LSA to merge software repository identities. pages 592–595, 2012. 10.1109/ICSM.2012.6405329.

Henrik Leopold, Sergey Smirnov, and Jan Mendling. On the refactoring of activity labels in business process models. *Information Systems*, 37(5):443–459, 2012.

Sherlock A. Licorish and Stephen G. MacDonell. Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Inf. Softw. Technol.*, 56(12):1578–1596, 2014. ISSN 09505849. 10.1016/j.infsof.2014.02.004.

Haibing Lu, Yuan Hong, Yanjiang Yang, Lian Duan, and Nazia Badar. Towards user-oriented RBAC model. *J. Comput. Secur.*, 23(1):107–129, 2015. ISSN 0926227X. 10.3233/JCS-140519.

Walid Maalej and Hans-Jörg Happel. Can Development Work Describe Itself? *7th IEEE Work. Conf. Min. Softw. Repos. (MSR 2010)*, pages 191–200, 2010. 10.1109/MSR.2010.5463344.

Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. A Survey of Role Mining. *ACM Comput. Surv.*, 48(4):1–37, 2016. ISSN 03600300. 10.1145/2871148.

W. Poncin, A Serebrenik, and M. van den Brand. Process Mining Software Repositories. *2011 15th Eur. Conf. Softw. Maint. Reengineering*, pages 5–14, 2011. ISSN 1534-5351. 10.1109/CSMR.2011.5.

Vladimir A. Rubin, Christian W. Günther, Wil M. P. Van Der Aalst, Ekkart Kindler, Boudewijn F. Van Dongen, and Wilhelm Schäfer. Process mining framework for software processes. In *Softw. Process Dyn. Agil.*, volume 4470, pages 169–181. Springer, 2007. ISBN 978-3-540-72425-4. 10.1007/978-3-540-72426-1_15.

Stefan Schönig, Cristina Cabanillas, Stefan Jablonski, and Jan Mendling. Mining the Organisational Perspective in Agile Business Processes. In *BPMDS*, pages 37–52, 2015a. 10.1007/978-3-319-19237-6_3.

Stefan Schönig, Cristina Cabanillas, Stefan Jablonski, and Jan Mendling. Mining the Organisational Perspective in Agile Business Processes. In *BPMDS*, volume 214 of *LNBIP*, pages 37–52. Springer, 2015b. 10.1007/978-3-319-19237-6_3.

Minseok Song and Wil M P van der Aalst. Towards comprehensive support for organizational mining. *Decis. Support Syst.*, 46(1):300–317, 2008. ISSN 01679236. 10.1016/j.dss.2008.07.002.

B F van Dongen and Sh Shabani. Relational XES : Data Management for Process Mining. *BPM Cent. Rep. BPM-15-02*, 2015. URL `BPMcenter.org`.

H M W Verbeek, Joos C A M Buijs, Boudewijn F Van Dongen, and Wil M P Van Der Aalst. Xes, xesame, and prom 6. In *Inf. Syst. Evol.*, pages 60–75. Springer, 2010.

H. M W Verbeek, Joos C A M Buijs, Boudewijn F. Van Dongen, and Wil M P Van Der Aalst. XES, XESame, and ProM 6. In *Lect. Notes Bus. Inf. Process.*, volume 72 LNBIP, pages 60–75. Springer, 2011. ISBN 3642177212. 10.1007/978-3-642-17722-4_5.

Annie T T Ying and Martin P. Robillard. Developer profiles for recommendation systems. *Recomm. Syst. Softw. Eng.*, pages 199–222, 2014. 10.1007/978-3-642-45135-5_8.

Liguo Yu and Srini Ramaswamy. Mining CVS repositories to understand open-source project developer roles. In *Proc. - ICSE 2007 Work. Fourth Int. Work. Min. Softw. Repos. MSR 2007*, pages 7–10, 2007. ISBN 076952950X. 10.1109/MSR.2007.19.