Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZAI-MSC-2015-15

Masterarbeit

im Studiengang "Angewandte Informatik"

A Platform for the Integration of Repository Mining and Software Analytics through Big Data Technologies

Fabian Trautsch

am Lehrstuhl für

Software Engineering for Distributed Systems

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

23. September 2015

Georg-August-Universität Göttingen
Zentrum für Informatik

Goldschmidtstrasse 7
37077 Göttingen
Germany

Tel.	+49 (5 51) 39-1 44 14
Fax	+49 (5 51) 39-1 44 15
Email	office@cs.uni-goettingen.de
WWW	www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 23. September 2015

Master's Thesis

A Platform for the Integration of Repository Mining and Software Analytics through Big Data Technologies

Fabian Trautsch

September 23, 2015

Supervised by
Dr. Steffen Herbold
Software Engineering for Distributed Systems Group
Institute of Computer Science
Georg-August-University of Göttingen, Germany

Prof. Dr. Konrad Rieck
Computer Security Group
Institute of Computer Science
Georg-August-University of Göttingen, Germany

Abstract

Researchers of various research areas (e.g. defect prediction, software evolution, software simulation) analyse software projects to develop new ideas or test their assumptions by performing case studies. But to analyse software projects, two different steps need to be taken: (1) the mining of the project data, which includes pre-processing, calculation of metrics and synthesizing composite results and (2) performing the analysis on basis of the mined data. The process of analysing a software project is often divided into these two steps and therefore, different tooling is used to perform them (e.g. CVSAnaly for mining, WEKA[50] for analysis). Furthermore, the tooling for these steps is very versatile. This raises the problem, that performed studies are often not replicable. Therefore, the possibility of performing meta-analyses (analyse an analysis) is not given. Additionally, the mining and combination of information from different data sources (e.g. mailing lists and source code repositories) is a complex and error-prone task. Furthermore, there is an ever-growing need for performing the analysis of a large amount of data efficient and fast. Our solution for these problems is the development of a platform, which incorporates the mining and the analysis of software projects and present it with an easy to use web interface to the researcher. Furthermore, our platform is designed as a general-purpose platform and uses different big data technologies and frameworks (e.g. Apache Spark, Apache Hadoop) to make an efficient and fast analysis of the mined data possible. Additionally, this platform can be automatically deployed in a cloud infrastructure. We have used our platform for mining different projects and conducting defect prediction research on the mined data. The results show, that our platform is a useful tool for conducting software project research.

Contents

List of Figures

List of Tables

Listings

List of Acronyms

AM	Application Master
API	Application Programming Interface
ASF	Apache Software Foundation
AST	Abstract Syntax Tree
AWS	Amazon Web Services
BI	Business Intelligence
BSON	Binary JSON
CC	Cyclomatic Complexity
CPDP	Cross-Project Defect Prediction
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
CSV	Comma Separated Values
CVE	Common Vulnerabilities and Exposures
DAG	Directed Acyclic Graph
DIT	Depth of Inheritance Tree
DSL	Domain-Specific Language
EMF	Eclipse Modeling Framework
EOL	Epsilon Object Language
ETL	Epsilon Transformation Language
FN	False Negatives
FP	False Positives

GB	Gigabyte
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
IaaS	Infrastructure as a Service
ITS	Issue Tracking System
JAR	Java Archive
JS	Javascript
JSON	Javascript Object Notation
LAN	Local Area Network
MB	Megabyte
MCC	Matthews Correlation Coefficient
MPI	Message Passing Interface
MR	MapReduce
MSR	Mining Software Repositories
MVC	Model-View-Controller
NFS	Network File System
OO	Object Oriented
OSGi	Open Services Gateway Initiative
OSS	Open Source Software
PHP	PHP Hypertext Preprocessor
POJO	Plain Old Java Object
POSIX	Portable Operating System Interface
RAID	Redundant Array of Independent Disks
RAM	Random-Access Memory
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
TN	True Negatives
TP	True Positives

VCC	Vulnerability-Contributing Commit
VCS	Version Control System
VM	Virtual Machine
XMI	XML Metadata Interchange
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language
YARN	Yet Another Resource Negotiator

1. Introduction

Repository mining and software analytics became an important research area nowadays [53, 56, 60, 63]. Both fields are interconnected: The results of repository mining are often used to perform software analytics. The area of software analytics is broad. It includes, e.g., software evolution [25, 45], defect prediction [43, 53] or software process simulation [56]. Most of the work in these and related areas use Open Source Software (OSS) projects like Linux [45], Eclipse [68, 94] or Firefox [93] as case studies to evaluate their research results. The first step in conducting research on a software project is the mining of the project data. The mining process comprises of several steps with the goal to extract useful information from software and its related artifacts. This information can include basic facts about the artifacts like changes that have been made and software quality metrics like Object Oriented (OO) metrics [21]. The mining process is time intensive and fault-prone, because the aggregation of information from different sources, like Version Control Systems (VCSs) and Issue Tracking Systems (ITSs), and the usage of different tools for the manifold tasks make the merging of the results complicated. Data repositories like tera-PROMISE [69] and GitHub archive[1] collect data and facilitate research on common data sets, but do not provide means for the data analysis. On the other hand, projects like Bitergia[2] try to analyse the projects. But Bitergia provides only Business Intelligence (BI) about projects, is a closed-source solution and, moreover, does not support complex analysis, e.g., based on machine learning. Other platforms, such as CrossPare [54] focus only on analytics and completely omit the data mining.

After the mining process is completed, researchers want to derive knowledge out of the mined data. The analysis possibilities are versatile (e.g. defect prediction, software evolution) and time consuming, because of the huge amount of mined data. Additionally,

[1]See: `https://www.githubarchive.org/`.
[2]See: `http://bitergia.com/`.

to perform the mining and analysis of a project an infrastructure must be set up, which covers the needs of the researcher.

Another problem that remains till today is, that studies that were performed are often not replicable [67]. The use of proprietary data for the development of models (e.g. defect prediction models) and the usage of different tooling and platforms for mining and processing the data are two of many reasons for this. This raises the problem, that results of one analysis can not be used to improve the results of another. Therefore, researchers are forced to start over every time they do a new analysis and can not reuse results from other researchers [16].

All the mentioned problems raise the need for a platform, which incorporates the mining and the analysis of projects. Furthermore, to make the platform useful for a broad range of researchers, it should have certain characteristics: (1) It should be easy to use and to deploy, (2) allow scalable and flexible analytics, (3) be extensible with new approaches, and (4) allow analysis of different kinds of data collected from multiple sources. The platform needs to be scalable, so that it can process even large projects in a acceptable amount of time. Flexibility and extensibility are important, because researchers often want to improve or customize their tools, so that they completely fit their needs. The data collection is the fourth important part: The platform needs to collect a variety of information, so that it can be used for many different research areas and not only for specific areas like defect prediction or software evolution.

In this master's thesis, we introduce a platform, which has all the aforementioned characteristics. It enables researchers to automatically mine project data and provide them with powerful and scalable tooling for the analysis of the data. Our platform mines the complete history of a project and collects static source code metrics, change metrics, social metrics, as well as the diffs for each revision. It makes use of the DECENT Model [66] to aggregate information from different sources and uses modern big data technologies like Apache Hadoop [3], MongoDB[4], and Apache Spark[5] for a fast project analysis. By employing MongoDB as a state-of-the-art NoSQL database, we grow a repository with data about software projects, that can be exploited by researchers. Furthermore, we developed a web-

[3]See: `https://hadoop.apache.org/`.
[4]See: `https://www.mongodb.org/`.
[5]See: `https://spark.apache.org/`.

based Graphical User Interface (GUI), where new projects can be easily added, mined and analysed. The results of the analysis are also presented via the GUI, together with the mined projects. Through this, the reporting also becomes part of the platform, which is the precursor for putting the software analytics into action.

Additionally, we developed scripts for the automated deployment of our platform in a cloud, based on OpenStack[6]. This way, our platform can be installed in private clouds, to create private data repositories and analytics approaches, as well as in public clouds for the sharing of data and analytics.

Our main contributions are the following:

- Development of a cloud platform for scalable software analytics based on Apache Spark.
- A software mining approach that is integrated into our platform to update data and collect information about new projects.
- Combination of different data sources that allow to combine text mining with source code, change and social metrics.

The remainder of this thesis is structured as follows: In Chapter 2 the foundations of our work is explained. Chapter 3 covers the conceptual design of our platform. In this chapter, we describe the interaction of the different parts of the system. The actual implementation is illustrated in Chapter 4. We present several case studies in Chapter 5 to evaluate our platform and its usefulness for mining software project data and conducting defect prediction research. In Chapter 6 we illustrate our results and discuss it. Problems, drawbacks and threads to validity of our approach are illuminated in Chapter 7. Chapter 8 lists related work. We compare the approaches introduced in this chapter with our own and highlight the differences. Finally, in Chapter 9, we conclude this thesis and point out different possible improvements and research directions.

Parts of this thesis were used as a foundation for a paper. The resulting paper can be found in Appendix C.

[6]See: `https://www.openstack.org/`.

2. Foundations

In the following chapter the foundations for this masters thesis are described. First, a short overview of the the DECENT model is given (see Section 2.1). Additionally, we describe different technologies, that are used for big data nowadays and which we also use in our platform. This includes Apache Hadoop (see Section 2.2), Apache Spark (see Section 2.3), and MongoDB (see Section 2.4). We also explain the fundamental concepts of the Yii Framework[1] (see Section 2.5) in this chapter. The last section is about the deployment management. We shortly describe the deployment management tools Vagrant[2] (see Section 2.6.1) and Ansible[3] (see Section 2.6.2).

2.1. DECENT Model

The context in which mined information should be used is important. Many tools and methods, which are used for data extraction and application, are context-specific. These tools are hard to adapt to different circumstances, because of the tight coupling between the extraction process and the application [66].

Makedonski et al. [66] proposed a model-based software mining infrastructure to circumvent these problems. The framework relies on "homogeneous high-level domain-specific models of facts extracted from raw assets" [66]. These different domain-specific models are then combined and transformed into models, that are related to a certain assessment task [66]. These models make use of the Eclipse Modeling Framework (EMF). The EMF is used to build tools and applications based on structured data models. The

[1]See: `http://www.yiiframework.com/`.
[2]See: `https://www.vagrantup.com/`.
[3]See: `http://www.ansible.com/`.

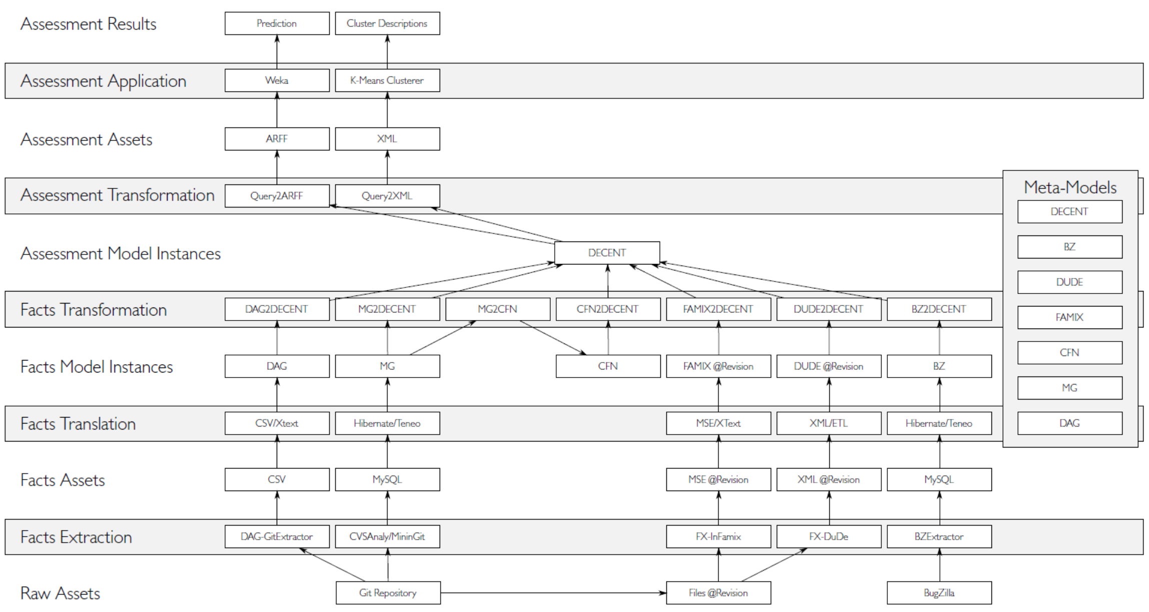


Figure 2.1.: Diagram for the DECENT domain-specific meta-model for developer-centric assessment tasks. Source: [66]

model, which represents (part of) a real world problem, needs to be specified in the XML Metadata Interchange (XMI) format. The EMF provides tools, like a code generation facility, which produce Java classes for the model. These classes can be used to interact with a model instance directly in Java code. Additionally, it produces different adapter classes, which enables viewing and editing the model [34].

Figure 2.1 illustrates a concrete instantiation of this model-based software mining infrastructure. DECENT is a meta-model for "developer-centric assessment tasks, including the corresponding concepts and the relationship among them" [66]. The mining process to collect the data, that is needed for a concrete instantiation of a DECENT model, is divided into different steps [66]:

1. **Facts Extraction:** Raw assets (like Git repositories, BugZilla reports) are used to extracts facts out of it. There are different tools used for this facts extraction process:

- **CVSAnalY**[4]: CVSAnaly is widely used in research [39, 55]. It processes the VCS logs and stores its extracted facts into a MySQL database. But CVSAnalY flattens the revision hierarchy. Therefore, a Directed Acyclic Graph (DAG) of the revision hierarchy is also extracted, "by means of a separate custom facts extractor (DAG-GitExtractor), which produces a Comma Separated Values (CSV) representation of the revision hierarchy" [66].
- **InFamix**[5]: Files within each revision of the project are processed by InFamix. InFamix processes Java or C/C++ code and calculates different metrics for each file, class, method, and function. For each revision, a Famix 3.0 model instance is created. These models are in the MSE[6] notation and store the results of this fact extraction step. Furthermore, the extractor is wrapped in a "custom facts extractor automator (FX)" [66], which is responsible for the checkout of each revision and the facts extraction execution.
- **DuDe**[7] [31]: Similar to InFamix, DuDe is also wrapped in the same FX. It is used for duplication detection and generates facts assets in Extensible Markup Language (XML) format for each revision.
- **BZExtractor**: Makedonski et al. [66] developed a custom BugZilla[8] extractor, which is used to extract information (facts) from BugZilla issue reports. The results are stored in a MySQL Database.

2. **Facts Translation:** The generated heterogeneous facts assets are translated into homogeneous facts model instances. The structure of the facts assets are described by a set of meta-models (e.g. MG, FAMIX), which the facts model instances conform.
3. **Facts Transformation:** The for the assessment task relevant parts of the different model instances are then transformed into a assessment-specific domain-model, which conforms to the DECENT meta-model. In this step, Makedonski et al. rely on the Epsilon Transformation Language (ETL). Therefore, different information from

[4]See: `http://github.com/MetricsGrimoire/CVSAnalY`.
[5]See: `https://www.intooitus.com/products/infamix`.
[6]See: `http://www.moosetechnology.org/docs/mse`.
[7]See: `http://www.inf.usi.ch/phd/wettel/dude.html`.
[8]See: `https://www.bugzilla.org/`.

different sources at different abstraction levels are stored in the DECENT model instance.

4. **Assessment Transformation and Application:** The newly created DECENT model instance, can now be "queried to produce different assessment assets, which are fed into the assessment application" [66] (e.g. Weka). Makedonski et al. rely on Epsilon Object Language (EOL) scripts to query the model instance.

2.2. Apache Hadoop

Apache Hadoop[9] is a framework used for distributed processing of large data sets. It is developed as a project of the Apache Software Foundation (ASF). Apache Hadoop is designed to run fail-safe on commodity hardware. Large problems are split into smaller chunks, which are then distributed in a cluster, which was previously created. This way, Apache Hadoop can react on hardware failures by reassigning the job to other nodes. Furthermore, it utilizes the hardware capabilities of all nodes [8].

2.2.1. MapReduce

MapReduce (MR) is a programming model, that emerged at Google [26]. There was a need for a programming model, which can be used to process large datasets for possibly versatile tasks. The developer should be able to concentrate on the application code, without having to bother with common problems like load balancing, data partitioning or failure handling. A MR program consists of a *map* and a *reduce* function. The *map* function has key/value pairs as input and produces new key/value pairs as output. Now, the library sorts this output by key and those pairs with the same key are passed to the *reduce* function. This function implements some action to merge the values for that key together. An example for the application of the MR paradigm is the word count problem. The result for this problem should be a list of words and how often they occur in a text document [26].

Listing 2.1 shows Pseudo-code for a solution to the above stated problem based on MR. The *map* function emits pairs of the word and the value one for each word in a line. Then,

[9]See: `https://hadoop.apache.org/`.

the *reduce* function gets pairs of word and *"1"* as input. These pairs are created, by aggregating the result of the *map* function by key. The reduce function iterates over the *"1"* for each key and adds them up to create the word count.

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
        Emit(AsString(result));
```

Listing 2.1: MapReduce WordCount in pseudo-code. Source: [26]

The execution phases are depicted in Figure 2.2. They strongly depend on the actual MR paradigm implementation and the underlying hardware. There are seven processing phases [26]:

1. First, the input file(s) are split into M smaller pieces of configured size by the MR framework. Each piece is send to a *map* job afterwards, as input.
2. One node is the master node. It assigns idle nodes different *map* tasks.
3. Each worker node processes their assigned input split, parses key/value pairs and passes them to the *map* function. The results (intermediate key/value pairs) are buffered in memory.
4. Periodically, these buffered pairs are partitioned and written to local disk. The master node receives the buffered pairs locations and passes this location to the *reduce* worker nodes.
5. Each *reduce* worker reads the data, which was assigned to them, and sorts it by key.

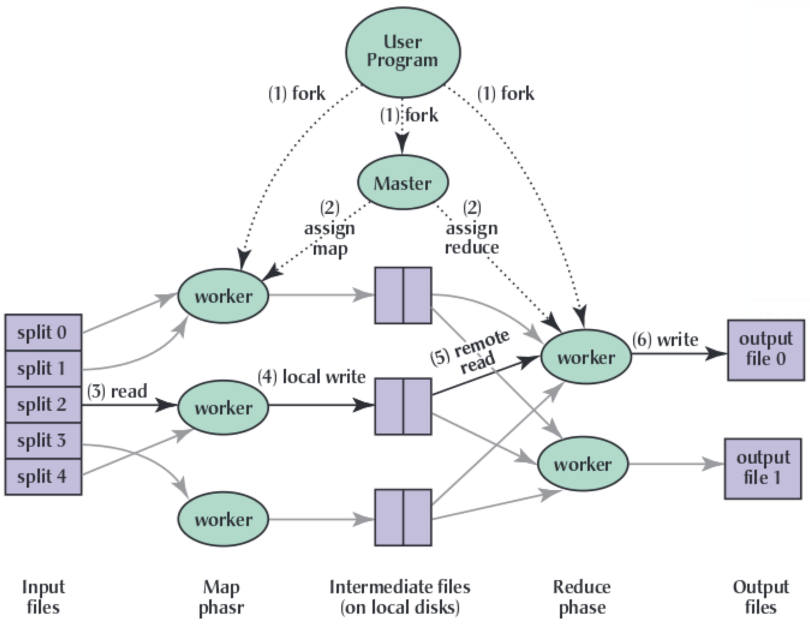


Figure 2.2.: MapReduce execution phases. Source: [26]

6. For each key in the input data the *reduce* function is executed and appends the result to the final output file.
7. After the completion of all *map* and *reduce* tasks, the master node wakes up the user program.

2.2.2. Hadoop Distributed File System (HDFS)

As we have seen in Section 2.2.1, all worker nodes of a Apache Hadoop cluster need access to shared data sets to perform the required actions. This raises the need for a distributed file system. A centralized file system does not fulfil the requirement of fail-safeness on commodity hardware. It would introduce a single point of failure. Hence, Apache Hadoop relies on its own file system, which is called Hadoop Distributed File System (HDFS). The HDFS was developed with the assumption, that hardware failures are the norm rather than the exception. Therefore, the recovery mechanism is implemented in software and Apache

Hadoop resigns on the usage of typical hardware mechanism like Redundant Array of Independent Disks (RAID) [9].

The HDFS architecture is illustrated in Figure 2.3. It employs a master/slave concept. Clients can perform read and write operations on the HDFS. An HDFS consists of different parts [9].

1. A single (primary) namenode: Master server, that manages the namespace of the file system and regulates access to files. It executes file system operations (open, close, rename of files and directories). Furthermore, it maps blocks to datanode and stores the filesystem metadata.
2. Datanodes (usually one per node in the cluster): Manages the storage of the nodes, where the datanode runs on. Responsible for processing read and write requests from the file system clients. Furthermore, it performs block operations (creation, deletion, replication), if the namenode gives the instruction. Furthermore, the blocks of a file are replicated to achieve fault tolerance.
3. A secondary namenode: Creates checkpoints periodically. The current image of the primary namenode is downloaded, additionally to the log files, which are edited and joined. The resulting new image is uploaded back to the primary namenode.

The HDFS is written in Java. Therefore, every machine that has Java support is able to run the namenode or datanode software.

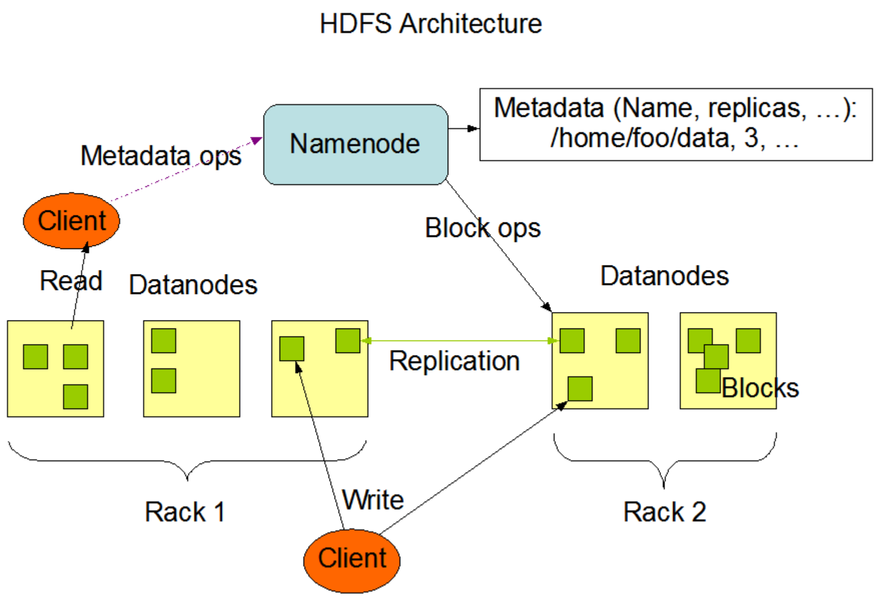


Figure 2.3.: HDFS architecture. Source: [9]

2.2.3. Yet Another Resource Negotiator (YARN)

Yet Another Resource Negotiator (YARN), which is also called MR 2.0 (MRv2) has been introduced in Apache Hadoop version 2.0. The reasons was, that "developer extended the MapReduce programming model beyond the capabilities of the cluster management substrate" [86]. The limitations of the MR Application Programming Interface (API) resulted in a misuse of it by developers [86]. Hence, the limitations were negotiated by not only focusing on MR as processing paradigm. Other processing paradigms can be added via user-defined applications. The architecture of YARN is depicted in Figure 2.4 [86].

The ResourceManager consist of a Scheduler and an ApplicationManager (AMService). Clients of the Apache Hadoop cluster can interact with the ResourceManager. The Scheduler is responsible for the allocation of resources for the different running applications. But it needs to subject to constrains like capacities or queues. The ApplicationManager handles job submissions and negotiates the first container [86].

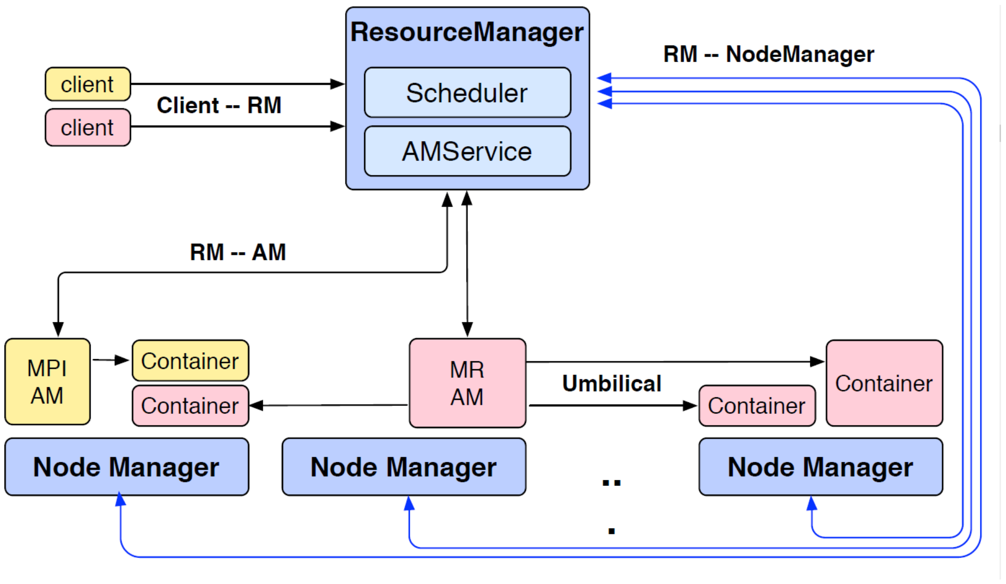


Figure 2.4.: YARN architecture (in blue the system components, and in yellow and pink two applications running). Source: [86]

The local resources on each node are monitored by the NodeManager, which runs on each node in the cluster. Furthermore, it reports faults and manages the container lifecycle (e.g. start, kill). Each application has an Application Master (AM), which negotiates resources from the Scheduler in form of containers. In Figure 2.4 there are two different applications. As we have stated above, the applications that run on YARN are not limited to the MR paradigm. Therefore, the yellow application uses the Message Passing Interface (MPI), whereas the pink application uses the classical MR. There are several nodes in the cluster, which run the NodeManager. The pink application has its AM on the second node, whereas the yellow application has its AM on the first node. Furthermore, the AMs are communicating with the ApplicationManager to get resources in form of containers. The container for both applications are located at the first node. The pink application additionally has resource container on the last depicted node. Furthermore, all NodeManagers are communicating with the ResourceManager and giving status reports to it [86].

2.3. Apache Spark

Apache Spark is a framework, implemented in Scala[10], which focuses on big data analytics. Because of MRs inefficiency for certain applications (especially multi-pass, which require "low-latency data sharing across multiple parallel operations" [12]), the AMPLab[11] at UC Berkeley developed this framework. These multi-pass applications are common in big data analytics: Examples are iterative algorithms like PageRank, interactive data mining, where data is loaded into the Random-Access Memory (RAM) across a cluster and repeatedly queried, and streaming applications, which maintain an aggregate state over time [91, 92].

The core of Apache Spark is an abstraction called Resilient Distributed Datasets (RDDs). This abstraction was especially designed to efficiently support the above named applications. RDDs are defined as a "fault-tolerant collection of elements that can be operated on in parallel" [10]. Furthermore, they have the ability to rebuild lost data, if something failed, using lineage. Listing 2.2 shows an Apache Spark text search example in Scala. This code is used to count lines, which contain errors in a large logfile, stored in the HDFS (see 2.2.2). At first, the file is loaded from the HDFS. Then it is filtered in a way, that it only contains lines with "ERROR" in it. Now we have a classical map reduce: The lines are mapped to key/value pairs with (line, 1) and after that reduced (the ones are summed up). This is similar to the word count example explained in Section 2.2.1.

```scala
val file = spark.textFile("hdfs://...")
val errs = file.filter(_.contains("ERROR"))
val ones = errs.map(_ => 1)
val count = ones.reduce(_+_)
```

Listing 2.2: Apache Spark text search example. Source: [92]

Figure 2.5 depicts the lineage of each RDD. Each object in the dataset has a pointer to its parent with the information about how the parent was transformed before. Therefore, if a failure occurs and an object is lost, it is possible to reconstruct the object using the lineage of it. There are different types of RDDs, but they only differ in the implementation of the RDD interface, which consists of three operations [92]:

[10]See: `http://www.scala-lang.org`.
[11]See: `https://amplab.cs.berkeley.edu/`.

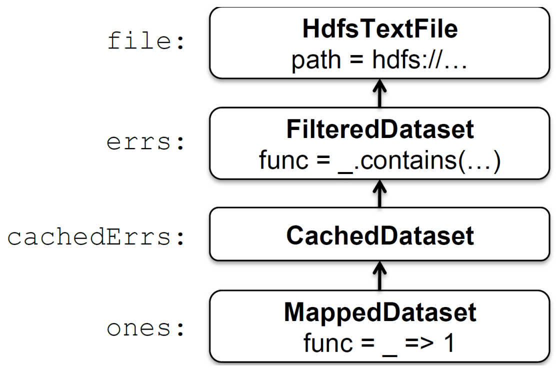


Figure 2.5.: Lineage chain for the distributed dataset objects defined in Listing 2.2. Source: [92]

1. getPartitions, return a list of partition IDs.
2. getIterator(partition), iterate over a partition.
3. getPreferredLocations(partition), used for task scheduling.

Figure 2.6 illustrates a comparison of the performance of a logistic regression job between Apache Spark and Apache Hadoop (see Section 2.2). Input was a 29 Gigabyte (GB) dataset and it was processed on 20 "m1.xlarge" EC2 nodes[12] with 4 cores each. Apache Spark outperforms Apache Hadoop, if more than one iteration is used. This highlights the main application area of Apache Spark.

Apache Sparks cluster mode architecture is depicted in Figure 2.7. Spark applications are coordinated by the SparkContext object in the main program (in Spark called: *driver program*) and run as an independent set of processes on a cluster. But to run a cluster, the SparkContext must be connected with a cluster manager. Hence, Apache Spark offers different connections or deployment methods [11].

1. **Standalone Mode**, where Apache Spark itself provides a simple cluster manager.
2. **Mesos Mode**, where Apache Mesos[13] is used as cluster manager.
3. **YARN Mode**, where YARN is used as cluster manager (see Section 2.2.3).

[12]See: `https://aws.amazon.com/de/ec2/previous-generation/`.
[13]See: `http://mesos.apache.org/`.

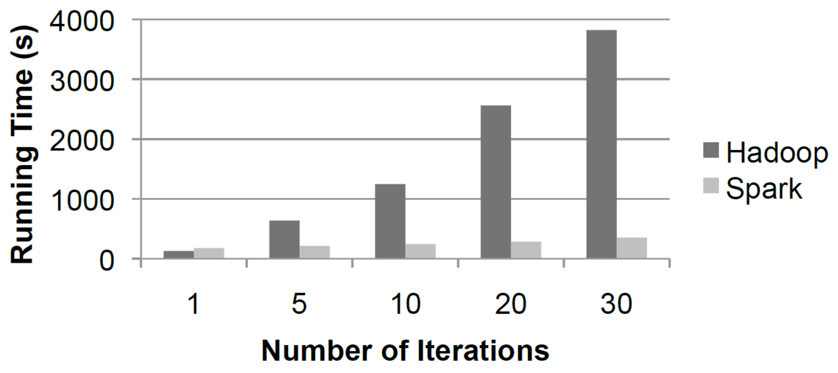


Figure 2.6.: Comparison of the logistic regression performance of Apache Hadoop and Apache Spark. Source: [92]

Additionally, Apache Spark provides Amazon EC2[14] scripts, which launch a standalone cluster on an Amazon EC2.

When Apache Spark is connected to the cluster manager, it acquires *executors* on nodes in the cluster. These have different tasks, including running computations and storing data for the executed application. After that, Spark sends the application code (e.g., a Java Archive (JAR) or Python file, which were passed to SparkContext), to the different *executors*. The last step is the distribution of tasks to the *executors*, which then run these tasks [11]. There are several things to note.

- Apache Spark spawns *executor* processes for each application, which run tasks in multiple threads and stay up for the whole application lifetime. Therefore, applications are isolated on the scheduling and executor side, but this has the downside, that no data can be shared between different applications.
- Apache Spark does not notice the cluster manager. The cluster manager only needs to answer on the acquisition of executor processes.
- The *driver program* needs to accept incoming connections from its executors. Furthermore, it must be listening for and accept these connections throughout its whole lifetime.

[14]See: `https://aws.amazon.com/de/ec2/`.

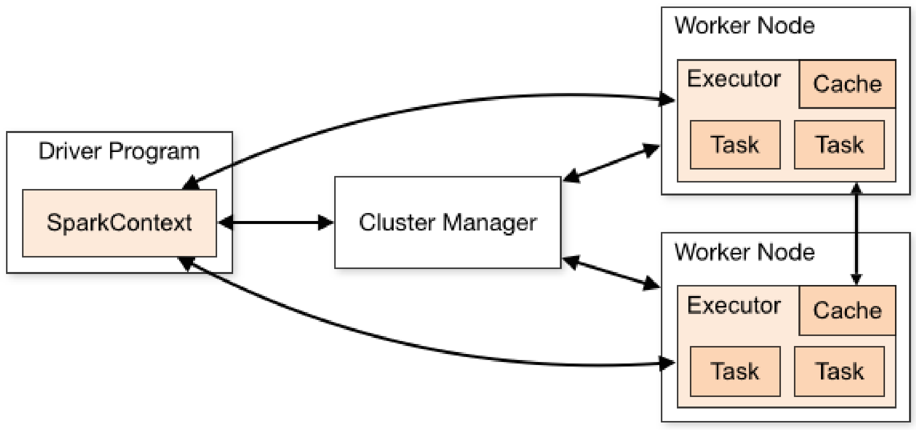


Figure 2.7.: Apache Spark cluster architecture overview. Source: [11]

- Because of the tight connection between the driver and the worker nodes (driver schedules tasks on the worker), they should run on the same Local Area Network (LAN) with a high-speed connection.

Furthermore, Apache Spark supports libraries for different applications [10]:

- **Machine Learning Library (MLlib)**[15], for machine learning tasks like clustering, classification, regression, and dimensionality reduction.
- **GraphX**[16], for graph and graph-parallel computation.
- **SparkR**[17], a package that provides a frontend for using Apache Spark from R.
- **SparkSQL**[18], which provides DataFrames as programming abstraction and can act as a distributed SQL querying engine.

[15]See: `https://spark.apache.org/docs/latest/mllib-guide.html`.
[16]See: `https://spark.apache.org/docs/latest/graphx-programming-guide.html`.
[17]See: `https://spark.apache.org/docs/latest/sparkr.html`.
[18]See: `https://spark.apache.org/docs/latest/sql-programming-guide.html`.

2.4. MongoDB

MongoDB is an open source NoSQL database, which is document-oriented. It is commercially supported by 10gen and is developed in C++. The first public release was in 2009 and the actual version is 3.0.4 [4]. MongoDB is widely used in research and industry [41, 73, 87].

MongoDB organize its data in *documents* instead of classical tables with rows and columns. Each of these documents is an associative array, which can consist of scalar values, lists or even nested associative arrays. These documents are saved in collections, which can be seen as a group of documents. Naturally, MongoDB documents are serialized as Javascript Object Notation (JSON) objects. In fact, they are stored in the Binary JSON (BSON) format, which are binary encoded JSON objects. Another feature of MongoDB, which is similar to a relational database, is indexing. Each document is identified by an "_id" field. By default, an unique index over this field is created. Indexing is important for efficiently querying data from the database, but can have a negative impact on write operations. Furthermore, a "compound index" can be specified, which is an index over several fields within a specified collection [4].

MongoDB posses two features, which are especially interesting in a big data context: Durability and concurrency. Durability is achieved by the creation of replicas. MongoDB uses a Master-Slave replication mechanism. Therefore, the master can write and read files and the slaves serve as a backup. Hence, slaves can only perform reading operations. If the master fails, the slave with the most recent data is promoted. MongoDB uses an asynchronous replication technique. Therefore, the updates that are done to the MongoDB are not spread immediately [4].

Concurrency is archived by sharding. It is used to scale the performance on a cluster of servers. Sharding describes the process of splitting the data evenly across the cluster. Hence, a parallel access is possible [27]. The process of splitting up a collection is depicted in Figure 2.8. It reduces the number of operations that each shard needs to handle and the amount of data that each server needs to store. Therefore, it enables higher throughput and the use of large data sets, which possibly can not be stored on a single node [27].

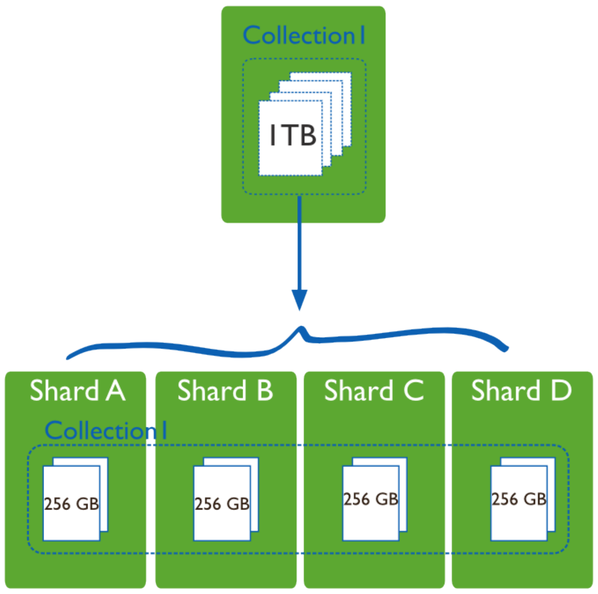


Figure 2.8.: MongoDB sharding overview. Source: [72]

Another advantage of using MongoDB as data storage backend is the high availability of tools, which support MongoDB. MongoDB is supported by common languages like Java, Python, PHP Hypertext Preprocessor (PHP), Perl and there also exists frameworks, which make the data management easier. One example for such a framework is Morphia[19], which is a Plain Old Java Object (POJO) mapper, that makes the usage of MongoDB in Java more comfortable [71]. Furthermore, there are also connectors to Apache Spark (see Section 2.3). Two examples are Spark-MongoDB[20] and Deep-Spark[21]. These connectors relieve the usage of Apache Spark with MongoDB.

[19]See: `https://mongodb.github.io/morphia/`.
[20]See: `https://github.com/Stratio/spark-mongodb`.
[21]See: `https://github.com/Stratio/deep-spark`.

2.5. Yii Framework

Yii is a PHP based web framework, which is currently in version 2.0. Since version 2.0, Yii requires PHP 5.4.0 or above to run. Yii implements the Model-View-Controller (MVC) design pattern. Furthermore, it facilitates the code organization based on this pattern [89].

Figure 2.9 illustrates structure of an application running with the Yii framework. Models represent rules, business logic as well as data. Views are the representation of models, like they are seen by the user. Controllers have the task to take input and convert it to different commands, which can be processed by models and views. Besides MVC, the following entities are shown in the Figure [90]:

- **Entry scripts**: These scripts are used for starting a request handling cycle. Furthermore, these are the PHP scripts, which are directly accessible by users.
- **Applications**: Applications are objects, which are globally accessible. They manage all app
- **Application components**: Application components provide different services for fulfilling requests. They are objects, which need to be registered with applications.
- **Modules**: Modules can contain complete MVC by themselves. Furthermore, they are self-contained packages.
- **Filters**: Filter code is invoked before and after the request handling of each request by the controller.
- **Widgets**: Widgets are used in views. They are reusable and can contain logic.
- **Asset bundle**: Assets are files, which may be referenced in a webpage (e.g., a Cascading Style Sheets (CSS) file, image, video or Javascript (JS) file).

Furthermore, the Yii framework is extensible. At the time of writing this thesis, there exist over 1900 extensions[22]. Additionally, it provides many features, which are best practice nowadays, like ActiveRecords (for relational and NoSQL databases), multi-tier caching support or support for a Representational State Transfer (REST)ful API development.

[22]See: `http://www.yiiframework.com/extensions/`.

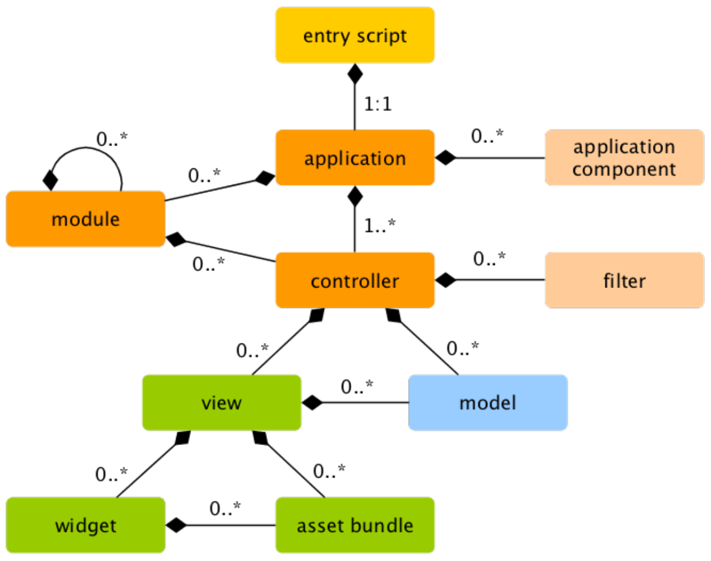


Figure 2.9.: Yii application structure overview. Source: [90]

2.6. Deployment Management

The setup of a cluster can be divided into two separate, but interconnected tasks: The infrastructure management and the software configuration management. To make these tasks manageable and executable by non-experts, several tools were developed in recent years. Two of these tools are described in this section: Vagrant[23] (see Section 2.6.1) for the infrastructure management and Ansible[24] (see Section 2.6.2)for the software configuration management.

2.6.1. Vagrant

As we use Vagrant in this thesis for deploying the infrastructure in a cloud environment, we refer to infrastructure management as software, that automates the provisioning of virtual machines. This provisioning is based on user-defined rules and can be understood as

[23]See: `https://www.vagrantup.com/`.
[24]See: `http://www.ansible.com/home`.

the instantiation of a Virtual Machine (VM) with an operation system. Therefore, only the VM with the operating system is set up by Vagrant, no other software is set up or configured, because this is handled by the software configuration management tool Ansible (see Section: 2.6.2).

As we have stated above, the setup of a cluster is a complex tasks. This derives from the complexity of cluster architectures and the need for testing these. Therefore, researchers need to have knowledge about cluster architectures, all the used software, the deployed infrastructure, and potentially also about the cloud platform used. Hence, they can not only focus on their research, but need to bother with the infrastructure where their research run on. This raises the need for tools like Vagrant: It allows the researcher to define the desired infrastructure in a file [74].

An example for a so-called Vagrantfile is shown in Listing 2.3. Vagrantfiles are written in Ruby [25] syntax. Furthermore, they allow the embedding of arbitrary Ruby code.

```
1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :

4  Vagrant.configure("2") do |config|
5    config.vm.define :server1 do |server1|
6      server1.vm.box = "precise64"
7    end

9    config.vm.define :server2 do |server2|
10     server2.vm.box = "precise64"
11   end
12 end
```

Listing 2.3: Vagrantfile example. Source: [74]

In this example, we configure vagrant with the API version 2 (line 4). We define two different virtual machines: Server1 and Server2 (see lines 5 and 9). The "x.vm.box" line indicates, which image should be installed on these machines. Here we have chosen "precise64" for both machines (see lines 6 and 10). It is not mentioned in this file, where the virtual machines should be created. Vagrant works with VirtualBox[26], by default. But there

[25]See: `https://www.ruby-lang.org/de/`.
[26]See: `https://www.virtualbox.org/`.

exist plugins, which add providers. Besides providers for VMware[27] and Amazon Web Services (AWS) [28], there is a provider for the OpenStack Infrastructure as a Service (IaaS) platform, which is used for this thesis[29]. As different providers need different input (e.g., the OpenStack provider requires login credentials) the example given above can not be applied to all providers. After the creation of the Vagrantfile, the developer can run the *vagrant up* command. Vagrant then communicates with the specified provider to provision the virtual machines defined in the file. Vagrant provides commands for the whole infrastructure life-cycle: Changes of the Vagrantfile are applied via *vagrant reload*, virtual machines are removed via *vagrant destroy*, the status of the infrastructure can be checked via *vagrant status* and the shut down of the virtual machines can be initiated via *vagrant halt* [51].

2.6.2. Ansible

As Vagrant (see Section 2.6.1) only covers the provisioning of the infrastructure, a tool is needed to deploy and configure the software on the different machines. Well-known tools for this task are Puppet[30], Chef[31], or Ansible. In this thesis Ansible is used. Nevertheless, they all have in common, that a Domain-Specific Language (DSL) is used to declare rules for the system setup. This approach, compared to manually configuring the systems, has the advantage, that it makes versioning possible. Furthermore, the infrastructure can be recreated at any time and the system configuration can be split into smaller pieces, which are then better manageable [6].

Listing 2.4 shows an example for an Ansible playbook. Playbooks are scripts that define which software should be installed in a VM and how the software should be configured. The YAML Ain't Markup Language (YAML) format is used for defining these playbooks, which ensures that the files are human-readable.

```
---
- hosts: webservers
```

[27]See: https://www.vmware.com/de.
[28]See: http://aws.amazon.com/de/.
[29]See: https://github.com/ggiamarchi/vagrant-openstack-provider.
[30]See: https://puppetlabs.com/.
[31]See: https://www.chef.io/chef/.

```
3   vars:
4     http_port: 80
5     max_clients: 200
6   remote_user: root
7   tasks:
8   - name: ensure apache is at the latest version
9     yum: pkg=httpd state=latest
10  - name: write the apache config file
11    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
12    notify:
13    - restart apache
14  - name: ensure apache is running (and enable it at boot)
15    service: name=httpd state=started enabled=yes
16  handlers:
17    - name: restart apache
18      service: name=httpd state=restarted
```

Listing 2.4: Ansible playbook example. Source: [7]

At first, it is defined which machines are targeted with this playbook (see line 2). Meaning, on which machines the playbook is executed. In the example above, the playbook is executed on the group "webservers". Furthermore, there are several variables defined: "http_port" (value: 80) and "max_clients" (value: 200) (see lines 3 - 5). These are environment variables for the execution of this playbook. After that, the user is chosen, with which privileges the playbook is executed (see line 6). Then several tasks are defined by giving them a descriptive name and an action (see lines 8-9, 10-13, 14-15). Ansible provides modules for different purposes[32], e.g., the service module for managing services. In the last step a handler is defined, which restarts the Apache httpd service using the service module (see lines 16 and 17). The calling of this handler is shown in the second task of this example (see line 13), where the Apache httpd service is restarted after changes were made at the configuration file.

[32]See: `https://docs.ansible.com/ansible/list_of_all_modules.html`.

3. Conceptual Design

In this chapter the design of our platform is illustrated. First, we give an analysis of today's situation of how software projects are analysed and highlight the existing problems (see Section 3.1). Then, we shortly describe our platform in Section 3.2. Additionally, we give an overview of how a project is mined (see Section 3.3) and analysed (see Section 3.4) with the help of the developed platform. In the last part of this chapter (see Section 3.5), we describe our cloud deployment.

3.1. Situation Analysis

There are several problems with researching OSS and software projects in general nowadays. These problems are unveiled, if we look at how research is done. The first step is the statement of a hypothesis. The basis of such a hypothesis, is often an idea or drawn from experience, former research or from the available research data. The next step is the formulation and execution of experiments to validate or reject the hypothesis. To perform this step, data is needed for the execution of such experiments. E.g., the field of software evolution mostly use OSS projects to perform their experiments [45, 68]. This is also true for the area of defect prediction [93, 94]. Data that is interesting for both areas are bug information data, change history or metrics of different software artifacts and is often obtained through Mining Software Repositories (MSR) [62]. Nevertheless, other research on software projects might use different data.

One problem is the mining of the data, which is a hard task, because of the huge amount of data and the need to combine data from different sources like VCSs and ITSs. Table 3.1 shows projects, which are often used in research and their number of commits and size in Megabyte (MB), the number of files, and example publications, where these projects are

Project	Number of Commits	Project Size in MB	Number of Files	Example Publications
Eclipse JDT Core	21618	162	7736	[68]
Apache Web Server	27265	280	3578	[13]
Apache POI	965	243	4378	[69, 79]
Apache log4j	2644	18	642	[69]
Apache xalan-j	1714	77	1331	[15, 69, 79]
Apache xerces-c	1169	50	1603	[69, 79]

Table 3.1.: Number of commits, files and size of popular OSS projects.

used in. This table shows, that if we have big projects like the Eclipse JDT Core, the mining is an exhausting and long running process. Nevertheless, there are also small projects or just one or more release version(s) of these projects used in actual research papers (see Table 3.1). Furthermore, the mining includes a pre- and postprocessing of the data. Otherwise, the data would not be in a usable format for further research. At the moment, researchers develop their own tools or using tools like CVSAnaly (see Section 2.1) for the mining process. The problem with this approach is a high variance in the quality of the mined data [13] and that experiments are often not replicable [67]. This raises the problem, that researchers can not base their own research on results from their colleagues. Furthermore, meta-analyses (analysing an analysis) are not possible.

Besides the mining problem, there is also the problem of analysing the data: Because of the huge size of the data, the analysis process is long-running and exhausting. Furthermore, some newly created methods need to return the analysis results (nearly) immediately, e.g., for informing the developer about problems with his commit [83]. This raises the need for an analysis platform, which can handle a huge amount of data and presents the results on-time.

Therefore, a mining and analysis platform needs to support parallel processing and it needs to be scalable. Furthermore, it should be able to mine software projects in a replicable way and it should gather as much data as possible, so that it can be used for a wide variety of different research areas. Furthermore, it should be easy to use and deploy, because most researchers are only interested in getting their analysis results and do not want to bother with setting up the infrastructure. Additionally, the platform should be extensible, because researchers might want to add other data sources. All these needs are fulfilled by the platform, which we present in the rest of this thesis.

3.2. Overview of the Platform

As it is explained in Section 3.1 there are two steps to perform to conduct research on a software project: (1) mining of the project data and (2) analysis of the mined data. Figure 3.1 illustrates an overview of the platform. In this figure, the logical software level is depicted, which is independent of the underlying infrastructure. The platform reduces the manifold tasks of the researcher to two tasks: First, the choosing of the project, which should be analysed. Second, the writing of the analysis program, which is used to create or gather the desired results. After the researcher has chosen a project, the mining process is started. The results are then saved in the MongoDB (see Section 2.4). After the mining process is completed, the researcher can write his program, which is based on Apache Spark (see Section 2.3). This analysis program is executed on the platform and accesses the mined data in the MongoDB. A more detailed view on the mining and analysis processes are presented in Section 4.1. All functionality is hidden behind a web front-end based on the Yii2 framework (see Section 2.5).

Figure 3.2 gives an overview of the architecture of the platform. Generally, the platforms infrastructure consists of four different parts: (1) webserver, (2) mining server, (3) MongoDB, and (4) Apache Hadoop cluster. The researcher only communicates directly with the webserver via a web front-end (see Section 4.2 for further information). The webserver accesses the MongoDB (e.g., displaying information about mined projects), the mining server (starting new mining jobs), and the Hadoop Cluster (via Apache Spark). The mining server is used to mine different projects and to save the results in the MongoDB. It is to note, that the MongoDB also should be deployed on a separate server for flexibility and scalability. The executed Apache Spark jobs on the Apache Hadoop cluster make use of the mined project data. Therefore, the Hadoop cluster needs to have access to the MongoDB.

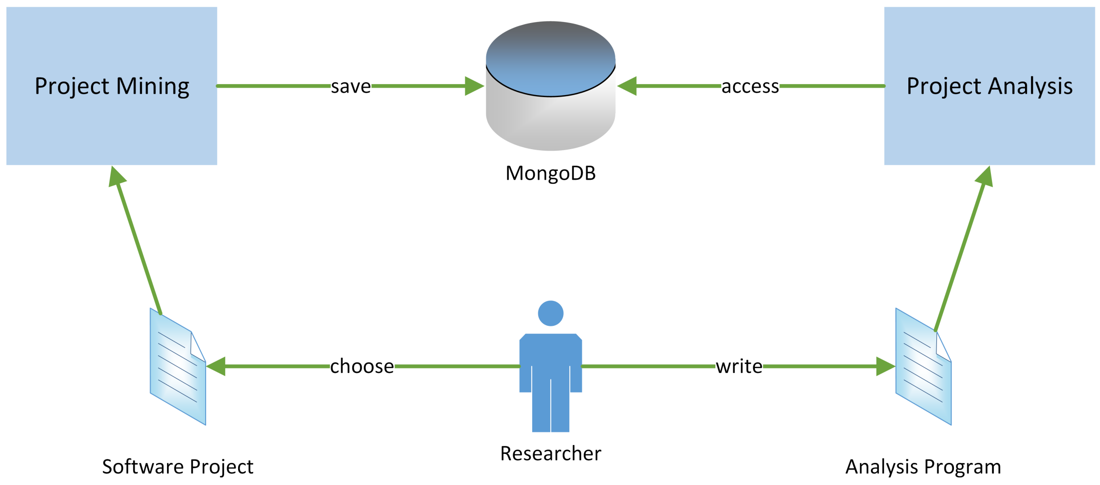


Figure 3.1.: Platform design - Software level. This figure shows the two main tasks for conducting research on a software project and how they are interconnected on the software level of the proposed platform.

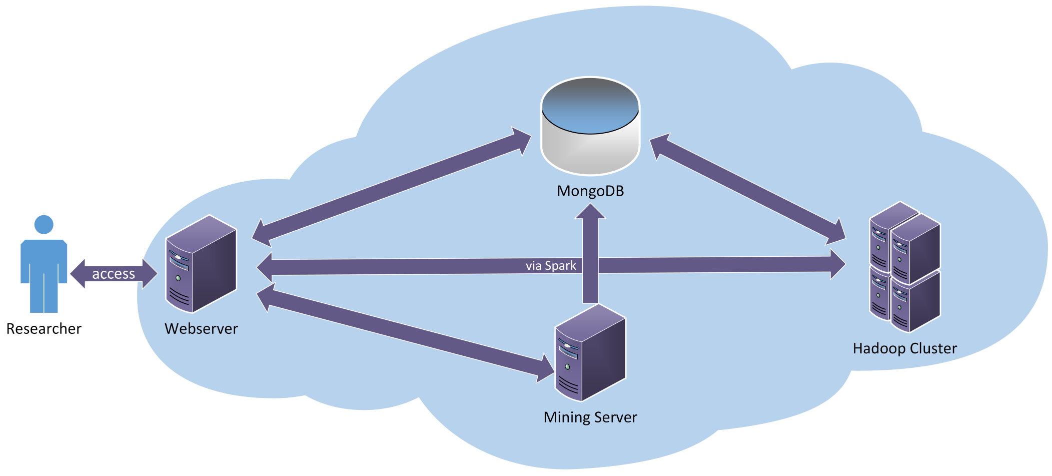


Figure 3.2.: Platform design - Architecture overview. This figure shows the architecture of the proposed platform, which consist of a webserver, a MongoDB, a Apache Hadoop cluster, and the mining server. The different parts of the Apache Hadoop cluster are not shown here. For further information, refer to Section 2.2.

3.3. Mining the Project - Make it ready-to-use

The mining of the project data is a hard task. In this section, we describe which data is mined (see Section 3.3.1). Furthermore, it is illustrated how the results are saved in the MongoDB and what schema is used (see Section 3.3.2). For a more detailed view on the mining process, see Section 4.1.

3.3.1. Project Processing

The first step in the project mining process is the collection of data from different sources. The next step is the calculation of intermediate results, which are based on the mined data. We base our project mining on DECENT (see Section 2.1). Figure 3.3 gives an overview of the different data, which is saved in the MongoDB. The data we obtain from the DECENT file are change-based. For each change (i.e. commit), we store all changed software artifacts and their location in the project. The software artifacts include files, classes, methods, functions as well as documentation files (e.g. READMEs).

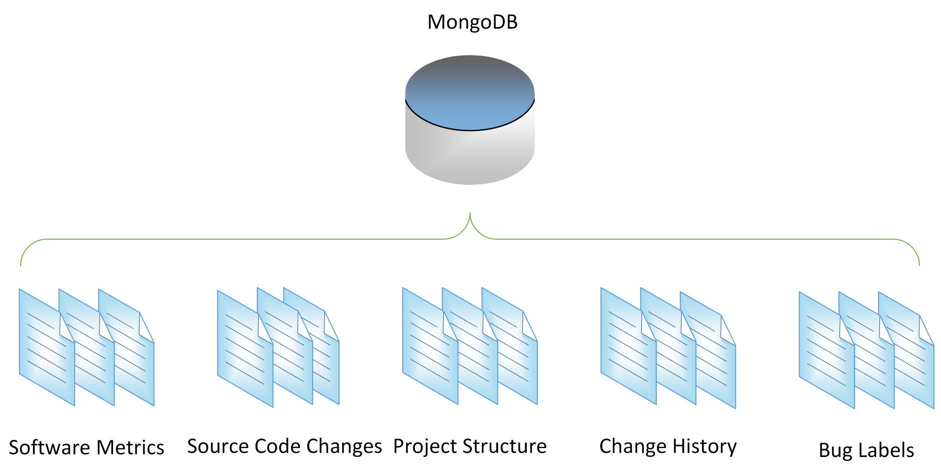


Figure 3.3.: Platform design - Saved data. This figure depicts the different data, which are saved in the MongoDB.

We store five different types of data for each artifact in the MongoDB. These are explained in the following.

1. **Software metrics**: This includes change metrics, social metrics (e.g. developer co-operation factor) and software quality metrics like Depth of Inheritance Tree (DIT) or Cyclomatic Complexity (CC). Furthermore, we save delta values for each metric, which indicate how the metric has changed since the artifact was last touched. If a value is zero for such a metric, it will not be saved in the MongoDB to save resources.
2. **Source code changes**: In addition to the metrics, we also save the concrete textual change that was made, i.e., the diff of the commit.
3. **Project structure**: For each change (i.e. commit) that is performed, we save the changed artifacts. Additionally, for each artifact, we save the location of it. Hence, it is possible to reconstruct the structure of the project at any point in time.
4. **Change history**: As it is stated above, we save what artifacts were changed in which commit and also how they were changed. Hence, it includes the changes in the software metrics, as well as changes in the source code (if we have code artifacts). This is important to note, because this makes text analysis of source code possible.
5. **Bug labels**: Bug labels and a confidence indicator label are saved for each artifact at each change. The confidence label indicates, if the resulting bug label (artifact is bug-prone or not) can be trusted.

3.3.2. Saving the Results

As we have stated above, the results of the project mining are stored in a MongoDB. We have chosen MongoDB as data storage backend, because it is already used in big data environments [73], it is scalable due to sharding (see Section 2.4), it enables the creation of replicas, has a good tool support and is easy to use and to set up.

Figure 3.4 depicts the MongoDB design we have chosen. Keep in mind, that MongoDB is not a relational database like MySQL. Therefore, concepts like primary or foreign keys or even a schema do not exist. Nevertheless, it is possible to link documents from different or the same collection(s) in MongoDB, by saving the object id in the referencing field. We used the Chen Database Notation [20] to visualize our design in Figure 3.4. But, we are fully aware of the fact, that this might be confusing, as MongoDB does not use schemas.

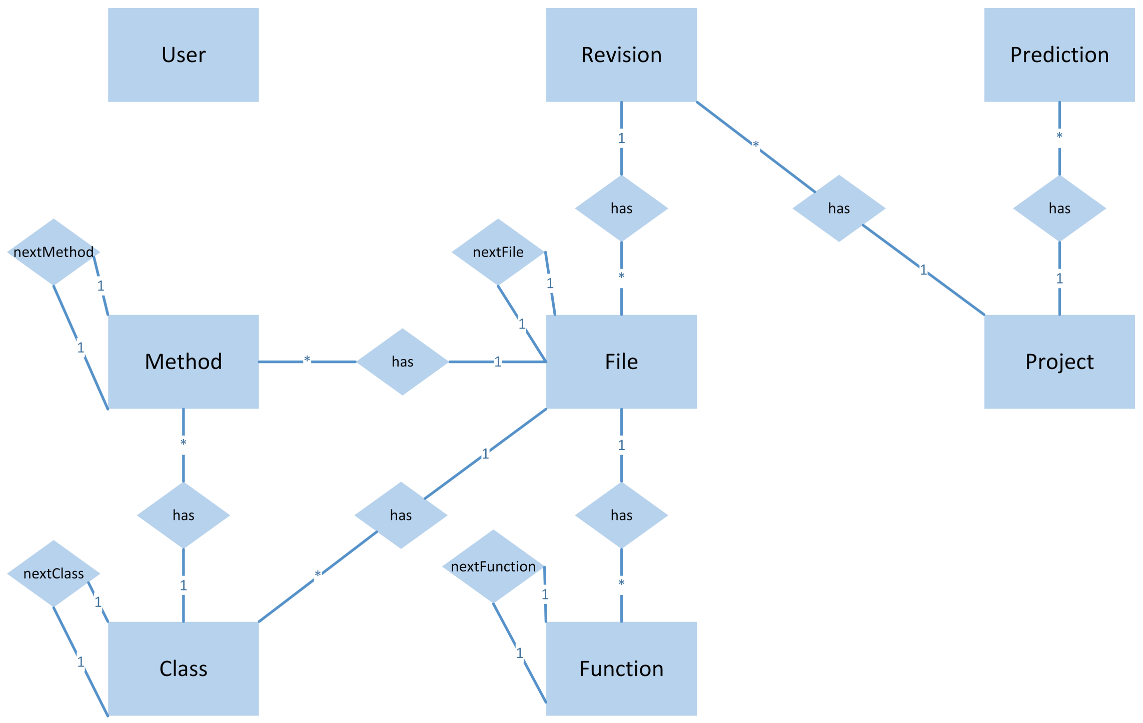


Figure 3.4.: Design of the used MongoDB. Chen Notation [20] is used, but there are no attributes displayed.

Nevertheless, it supports the explanation of the design of the database. Furthermore, we did not display attributes, because the figure would be unclear otherwise.

The user collection is a separate collection used by the web application for the user management. Additionally, every user is assigned a role, which is directly connected with the permissions for using the MongoDB (for further information refer to Section 4.2.3). The prediction collection is used in our case study to show, how intermediate results can be utilized by the webserver to create graphs or (in this case) evaluate the performance of a defect prediction classifier (see Chapter 5). A project can have multiple prediction collections, which must satisfy a specified scheme (for further information refer to Section 5.4). The rest of Figure 3.4 shows how the mined data is organized. A project has many revisions, which can have many files. Keep in mind, that the mined data is changed-based, which means, that we save file states instead of files. Hence, different file states from the same file are saved in this collection. Nevertheless, they all refer to another revision. The same holds true for functions, methods and classes. Important to note is, that we save a reference to the next state of the artifact in the current state of the artifact. Therefore, we can trace the evolution of an artifact easily, by following these next references, till there is no reference set. Furthermore, it is important to note, that files can have methods, functions, or classes.

3.4. Analyze the Project - Make Experiments to Gain Insights

Out of the data we mine, we can answer questions, which typically arise if we examine the evolution of a software project. E.g., how many files were changed from commit 2 to commit 70? Which files were moved in this time frame? Is the ownership of file X changing? When does it change? How is the change coupling of class X? Furthermore, this data can be used to conduct much more complex analysis, e.g., defect prediction research. An example for this application is given in Chapter 5.

After the mining, the analysis of the mined data is performed. Java, Python or R are often used languages in software project research. Therefore, our platform has the goal to support these. Unfortunately, at the time of writing this thesis, the platform supports only Java and Python. To perform analytics on our platform, the researcher first writes their analysis program. Spark allows local testing of jobs without the need to run them on the actual Hadoop cluster. Once the testing is completed, the researcher can upload and execute the JAR or Python file to the platform using the web front-end. For the execution, the Spark jobs are submitted to the Hadoop cluster, distributed among the nodes in the cluster and processed in parallel. The Apache Hadoop cluster is communicating with the MongoDB to gather all the needed mined data. It is possible to save the results in the HDFS (see Section 2.2.2), which is then read out by the webserver. The saved results are presented via the web-based GUI.

3.5. Cloud Deployment

As we mentioned above, both steps should be fast: the mining as well as the analysis step. But especially the analysis step is time-critical, e.g., for a real time response on commit [83].

Our platform is developed as a cloud platform. The aim was to develop it in a way, that the deployment is as easy as possible. Furthermore, the deployment model is adaptive, to allow changes based on the available computational resources. Therefore, we chose DevOps technologies to provide scripts that fully automate this process. As described in Section 2.6, the process is twofold: First, VMs have to be created and provisioned in our cloud. Second, the created machines need to be configured.

We need two different inputs to execute these tasks: User credentials and machine specifications. The user credentials are given by the cloud provider. In our case, we used a private cloud based on OpenStack. Table 3.2 shows the different parameter that need to be configured by the user.

Furthermore, we need to specify, how the virtual machines are provisioned. This includes information about the node count, what kind of flavour the node should have and also what access key is used to control the remote access to these machines. The flavour defines the computational capabilities of the nodes (e.g., number of Central Processing Unit (CPU)s, RAM and disk size) and is , in our case, defined in OpenStack.

Parameter	**Purpose**
Username	Identifier for the user, e.g. email address
Password	Secret key only known to the user
Tenant	Project identifier
Node Count	Number of virtual machines
Node Flavor	Computational capabilities of each node
Access Key	Access key for remote control of the nodes

Table 3.2.: This Table shows the parameter, which the user needs to specify to use the provided cloud deployment scripts. Based on: [48]

After the specification of these parameters, the user is able to call our Vagrant (see Section 2.6.1) and Ansible (see Section 2.6.2) scripts to deploy our platform in a cloud environment. Hence, the configuration effort is minimal. These scripts create the infrastructure explained in Section 3.2. Generally, the platforms infrastructure consists of four different parts: (1) webserver, (2) mining server, (3) MongoDB, and (4) Apache Hadoop cluster. However, all these parts can and should be separated from each other for more efficiency and flexibility. For further information about the infrastructure and its configuration refer to Section 4.5.

4. Implementation

The previous chapter introduced the conceptual design of our platform from a high-level perspective without technical details. This chapter presents these details for each part of the system. First, we give a detailed overview of our platform in Section 4.1. Section 4.2 describes the web-based GUI of our platform with all its possibilities, like the project management features (see Section 4.2.1), project analysis features (see Section 4.2.2), user management (see Section 4.2.3) and additional functionalities (see Section 4.2.4). Furthermore, we illustrate in detail how projects are mined in Section 4.3, which includes the creation of a DECENT model instance (see Section 4.3.1) and the writing of it into the MongoDB (see Section 4.3.2). Additionally, we describe how the mined data is utilized to perform analysis (see Section 4.4). In the last chapter, we present the technical details for the cloud deployment using Vagrant and Ansible (see Section 4.5).

4.1. Platform Description

Figure 4.1 illustrates the platform with the technical details. Nevertheless, not all details fit in this figure. Furthermore, we do not show the infrastructure here, only the connection of the different used tools. Figure 4.1 shows the two basic tasks we have described earlier in Chapter 3. On the one hand, the project mining and on the other hand the project analysis. The researcher can choose, which task should be performed with our platform. Nevertheless, if the researcher wants to analyse a project, the data must be mined before. The mining process starts with the setup of the project. The platform only requires the name, location to the git[1] repository and used programming language of the project. At the time of writing this thesis, the platform only supports git as VCS. Furthermore, it

[1]See: `https://git-scm.com/`.

only supports Java, C, and C++ as programming languages, because the InFamix tool (see Section 2.1) is able to parse only these languages. After the project is set up, it can be mined. At first, the git repository is cloned. Then, different tools are used to extract facts from this repository: CVSAnaly, FX-DuDe and FX-InFamix. These tools are introduced in Section 2.1. The figure illustrates, which technologies are used to transform the results of the different tools to EMF model instances. The results of CVSAnaly are processed with Xtext[2], Hibernate/Teeneo[3] and ETL. FX-InFamix and FX-DuDe are working on revision level. Therefore, each revision needs to be cloned. The results of FX-InFamix are processed via Xtext and the results of FX-DuDe with ETL. The intermediate EMF model instances are shown in the figure: MG, DAG and CFN are created out of the processing of the CVSAnaly results, whereas FAMIX model instances are created with the processing of the FX-InFamix results and the DUDE model instances with the processing of the FX-DuDe results. They are all transformed and connected with each other via ETL. The result is the DECENT model instance (see Section 2.1). The DECENT model is further processed via EOL: Another EMF model instance is generated, which is called DECENTMongo. As the last step, the DECENTMongo model instance is processed via Morphia (see Section 2.4). Hence, the results are written in the MongoDB.

After the project was mined, the researcher can execute the second basic task: The project analysis. For this step, a JAR or Python file needs to be uploaded via the web interface. In the background Apache Spark is used, which runs the analysis on a Apache Hadoop cluster. The results of the mining can be used via Apache Spark and intermediate results can be written in the MongoDB. Each part is described in more detail later in this chapter.

[2]See: `https://eclipse.org/Xtext/`.

[3]See: `https://wiki.eclipse.org/Teneo/Hibernate`.

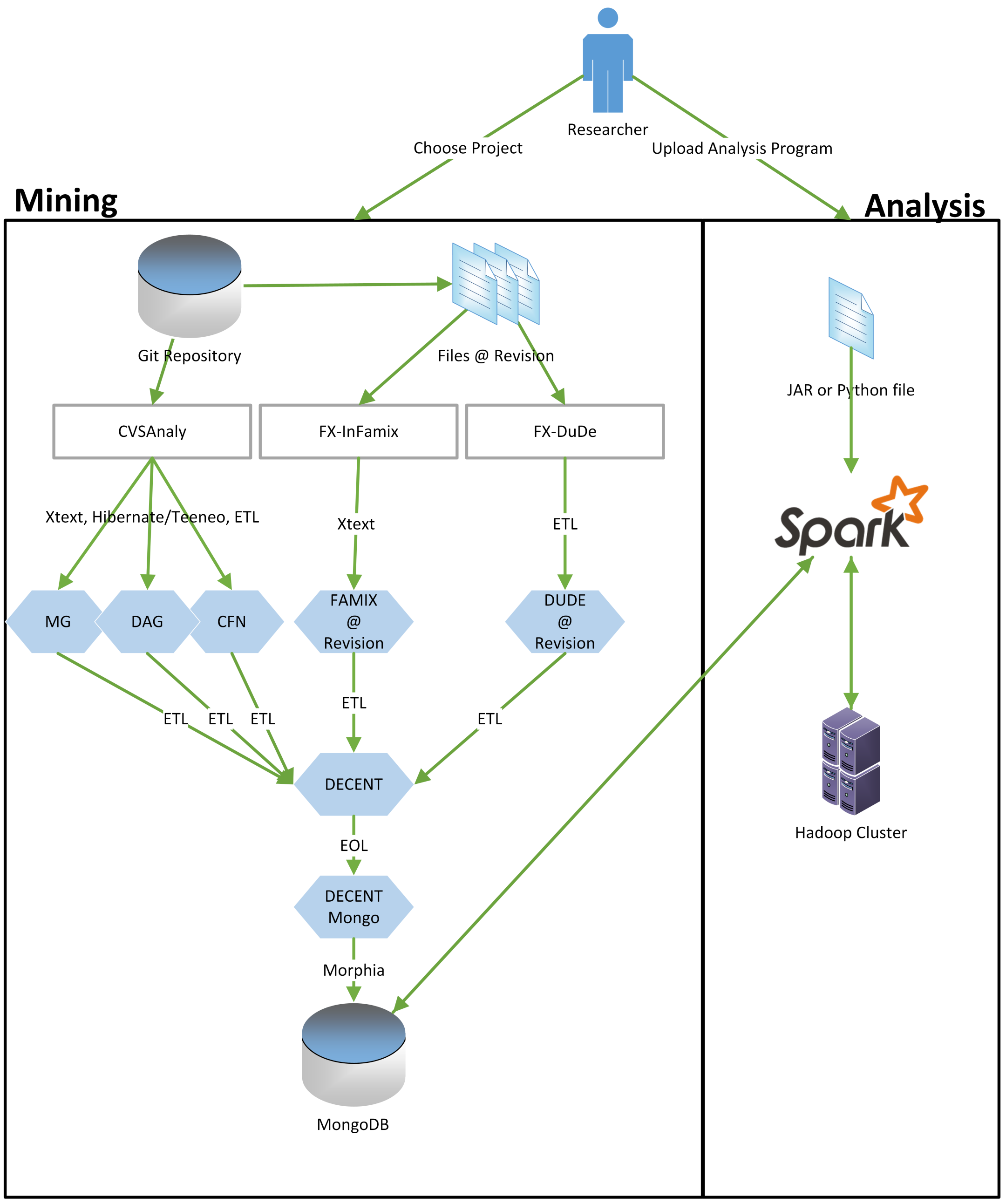


Figure 4.1.: Detailed overview of the platform. The infrastructure is not depicted. The focus lies on the connection of different system parts. Rectangles indicate programs we use, rhombuses are different EMF models, that are created. The @Revision shows, that these model instances are generated for each revision. The arrow labels show what tooling and technology is used to create the EMF model instances.

4.2. Graphical User Interface with Apache Spark Connection

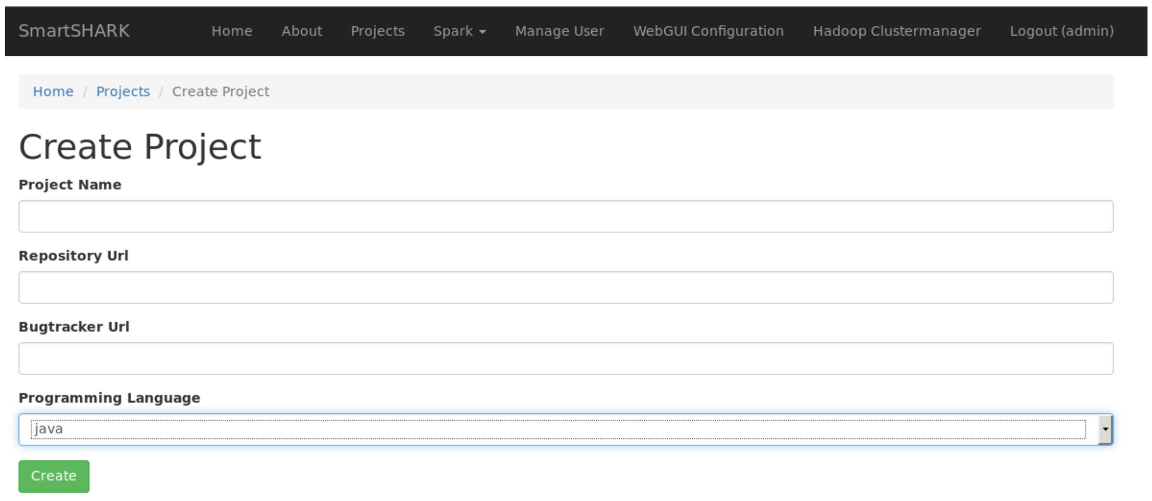
This section describes the GUI, which has an Apache Spark connection. From an end-user perspective, the web front-end is the central part of our platform. It is the only place where users directly interact with the platform and the starting point for both, the mining and the software analytics. We decided to build a web based GUI based on the Yii2 framework, because of the good implementation of the MVC design pattern, which allows easy extension of applications. Furthermore, we use MongoDB (see Section 2.4) for user authentication. The website runs on an Apache and at least PHP 5.4 is required. The decision to deploy our platform online and provide users with a web front-end also resolves the problem of distributing our platform, because interested users can simply visit the home page and do not need to download and install a standalone application.

The different project management features are explained in Section 4.2.1. Project analysis features are illustrated in Section 4.2.2. The user management is described in Section 4.2.3 and additional functionalities of the GUI are illustrated in Section 4.2.4.

4.2.1. Project Management Features

As described in Section 4.1 the projects need to be set up first. Figure 4.2 shows the interface for adding a new project. The platform needs a minimum amount of information to start the mining process, like the name, repository url and programming language. The bugtracker location is not used at the moment. Further restrictions are explained in Section 4.1. The actions, which are triggered by adding and mining a project, are explained in detail in Section 4.3.

The platform is able to list all added projects, together with detailed information about them like the mining progress. Figure 4.3 shows the list of projects, which the platform offers. There, information about every project (e.g., repository url, when was it last pulled) is displayed. Furthermore, the user can perform different actions for each project, like delete the project (bin symbol), mine the data (arrow-down symbol) or show defect prediction data (picture and arrow symbols) (further information about this feature is given in Section 5.4). Additionally, there is the possibility to sort the grid and search for specific information. Furthermore, we are able to backup projects, as well as show done backups



Create Project

Project Name

Repository Url

Bugtracker Url

Programming Language

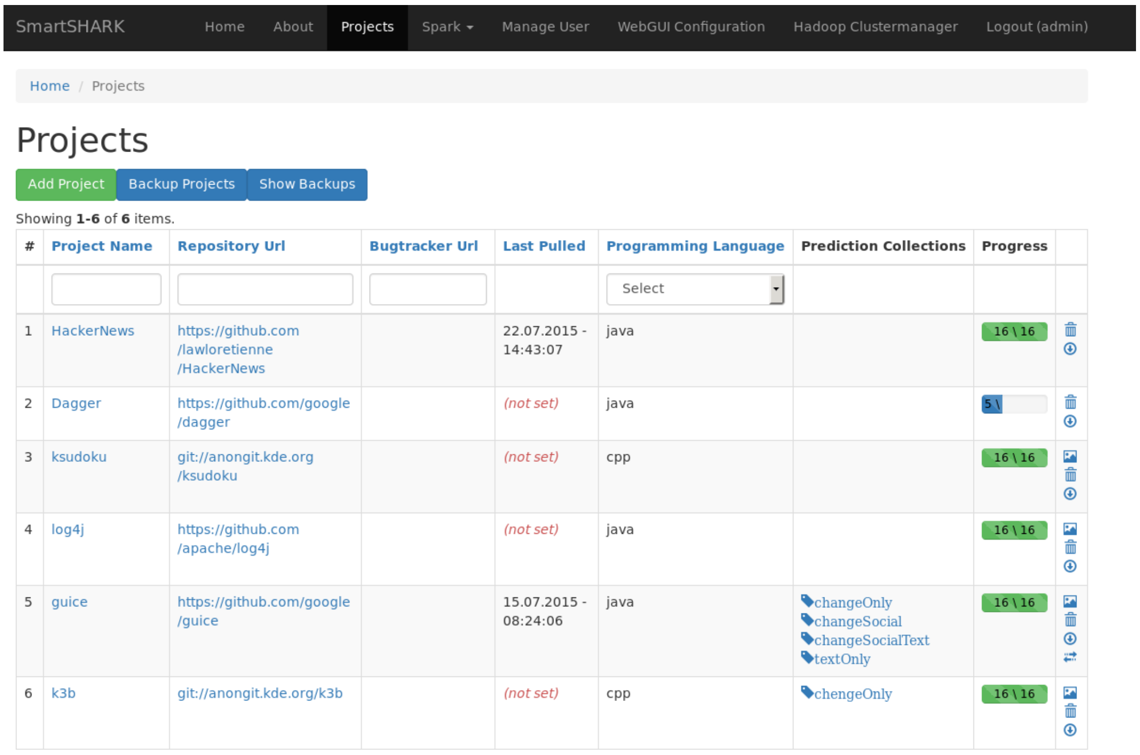
java

Figure 4.2.: GUI - Adding a project. The figure shows, what information is needed to add a project in the platform. This includes the name, repository url and the programming language. The bugtracker url is not used at the moment.

to download it later on. Further information about the backup functionality is given in Section 4.2.4. The last pulled attribute is not set at certain projects, because these projects were imported using a DECENT importer. This tool is explained in detail in Section 4.3.1.

Furthermore, there is the possibility to view a project in detail. Figure 4.4 shows an excerpt of the detailed view on a project. In this case, the project is k3b[4]. The interface gives us the possibility to download log files, or even configure the project using the text windows at the bottom of the page. For each step of the project processing (see Section 4.1) there exist configuration possibilities.

[4]See: `http://www.k3b.org/`.

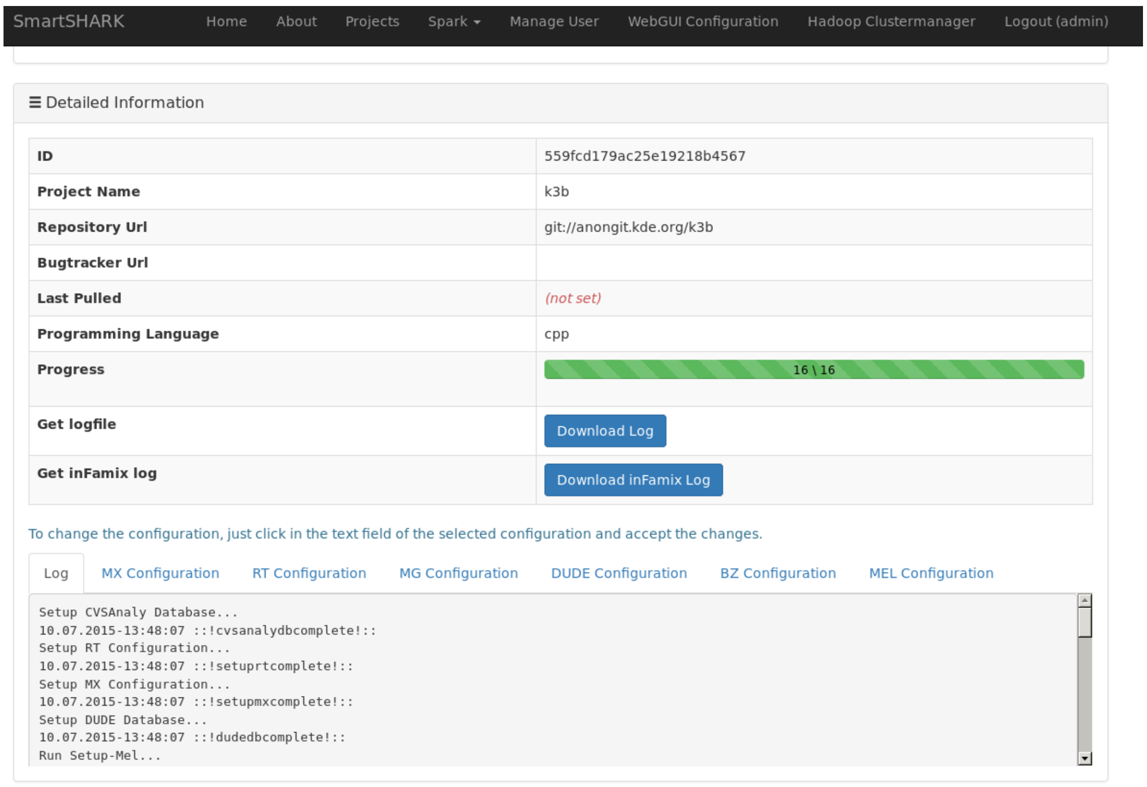


Projects

Showing **1-6** of **6** items.

#	Project Name	Repository Url	Bugtracker Url	Last Pulled	Programming Language	Prediction Collections	Progress	
1	HackerNews	https://github.com/lawloretienne/HackerNews		22.07.2015 - 14:43:07	java		16 \ 16	
2	Dagger	https://github.com/google/dagger		*(not set)*	java		5 \	
3	ksudoku	git://anongit.kde.org/ksudoku		*(not set)*	cpp		16 \ 16	
4	log4j	https://github.com/apache/log4j		*(not set)*	java		16 \ 16	
5	guice	https://github.com/google/guice		15.07.2015 - 08:24:06	java	changeOnly changeSocial changeSocialText textOnly	16 \ 16	
6	k3b	git://anongit.kde.org/k3b		*(not set)*	cpp	chengeOnly	16 \ 16	

Figure 4.3.: GUI - List of projects. The figure shows, how projects are displayed in the platform. Furthermore, it depicts the different available actions (delete, mine, defect prediction).



ID	559fcd179ac25e19218b4567
Project Name	k3b
Repository Url	git://anongit.kde.org/k3b
Bugtracker Url	
Last Pulled	*(not set)*
Programming Language	cpp
Progress	16 \ 16
Get logfile	Download Log
Get inFamix log	Download inFamix Log

To change the configuration, just click in the text field of the selected configuration and accept the changes.

```
Setup CVSAnaly Database...
10.07.2015-13:48:07 ::!cvsanalydbcomplete!::
Setup RT Configuration...
10.07.2015-13:48:07 ::!setuprtcomplete!::
Setup MX Configuration...
10.07.2015-13:48:07 ::!setupmxcomplete!::
Setup DUDE Database...
10.07.2015-13:48:07 ::!dudedbcomplete!::
Run Setup-Mel...
```

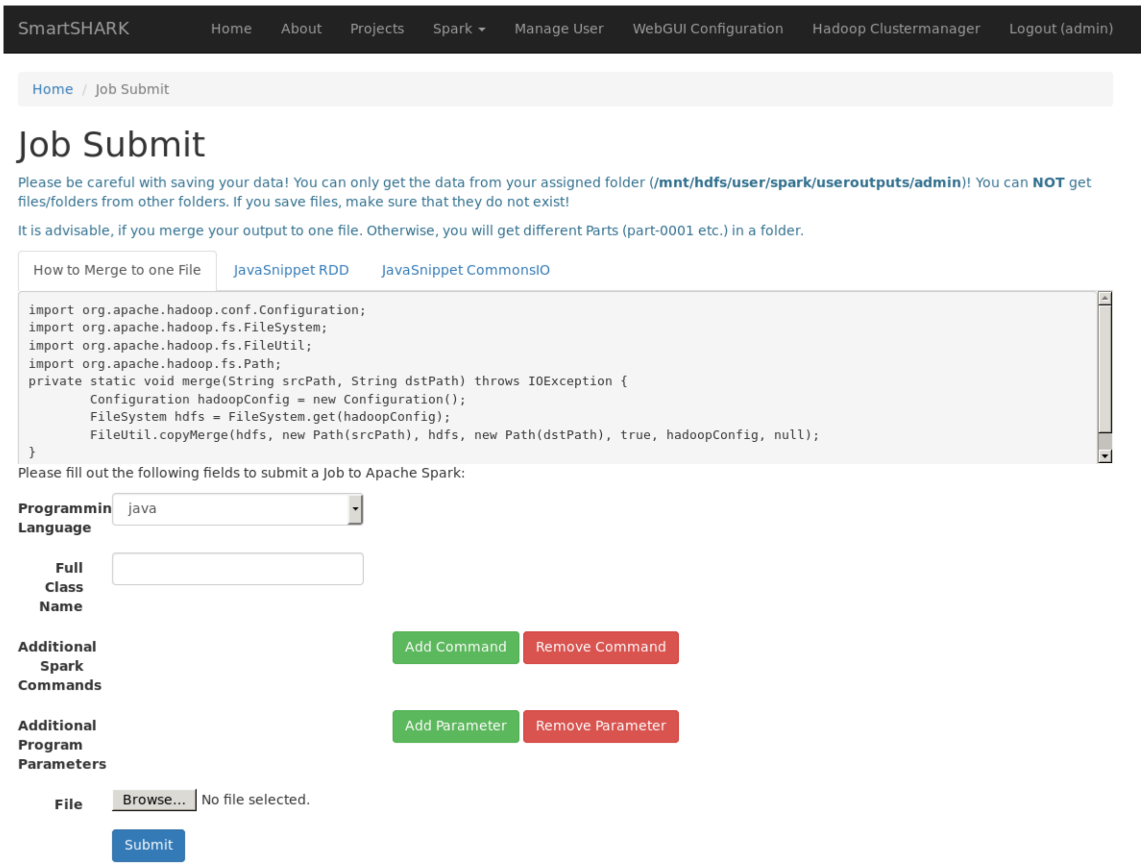
Figure 4.4.: GUI - Project detailed view. The figure shows, a detailed view of a project (in this case: k3b). There is the possibility to configure each project by themselves over the web interface using the text windows, depicted at the bottom of the figure. Furthermore, we get a detailed view of the project and the possibility to download log files, shown at the top of the figure.

4.2.2. Project Analysis Features

The second step, after mining the software project, is the project analysis. After the researcher wrote his analysis program (JAR or Python file), this program can be uploaded via the web interface. Figure 4.5 shows the user interface for this action. The researcher only needs to chose, if a JAR or Python file is uploaded. This must be specified, because if a JAR file is uploaded, the platform needs to know the full class name, which should be executed (e.g. de.smadap.wordcount.Run). Furthermore, the user can add additional Apache Spark commands (e.g. number of executors) and additional program parameters, that his uploaded program might need. What actions the upload progress triggers, is explained in detail in Section 4.4.

After the analysis program, based on Apache Spark, was uploaded, the progress and other information can be retrieved via the web interface. Figure 4.6 illustrates, how this information is displayed to the user. The most relevant information is displayed in a grid view. Additional information can be retrieved, by clicking on the eye on the left side. Furthermore, we can sort the grid and search for specific information and there is the possibility to terminate an application via the web interface.

After the Spark job finished, the results can be retrieved. Figure 4.7 shows, how the available results are displayed to the user. Each user has his own results directory. Everything that is saved in the corresponding folder is displayed on this page. Hence, it does not matter if the user saves files or whole folders. The user can chose what results to download or delete. Further information is given in Section 4.4.



Job Submit

Please be careful with saving your data! You can only get the data from your assigned folder (**/mnt/hdfs/user/spark/useroutputs/admin**)! You can **NOT** get files/folders from other folders. If you save files, make sure that they do not exist!

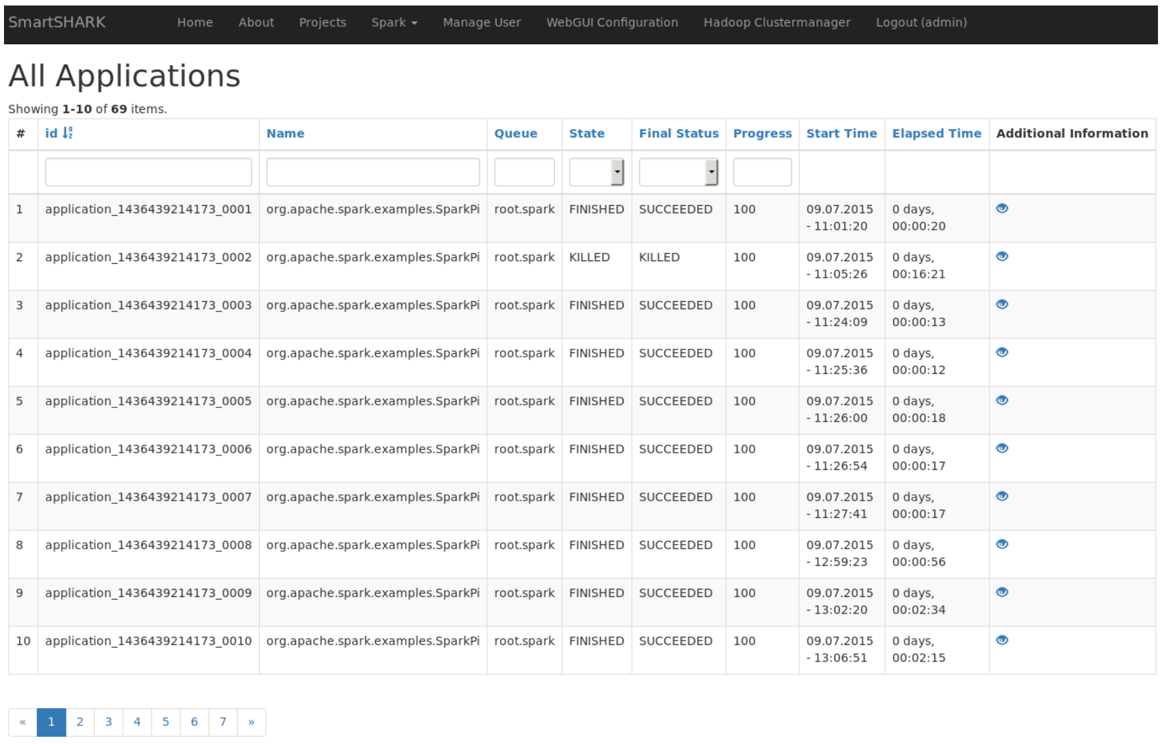
It is advisable, if you merge your output to one file. Otherwise, you will get different Parts (part-0001 etc.) in a folder.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileUtil;
import org.apache.hadoop.fs.Path;
private static void merge(String srcPath, String dstPath) throws IOException {
        Configuration hadoopConfig = new Configuration();
        FileSystem hdfs = FileSystem.get(hadoopConfig);
        FileUtil.copyMerge(hdfs, new Path(srcPath), hdfs, new Path(dstPath), true, hadoopConfig, null);
}
```

Please fill out the following fields to submit a Job to Apache Spark:

Programmin Language	java
Full Class Name	
Additional Spark Commands	
Additional Program Parameters	
File	No file selected.

Figure 4.5.: GUI - Spark submit. The figure shows, how Spark analysis programs are uploaded via the platforms web interface. The user needs to specify, if a JAR or Python file is uploaded. Furthermore, the user can add additional Apache Spark commands and program parameters.



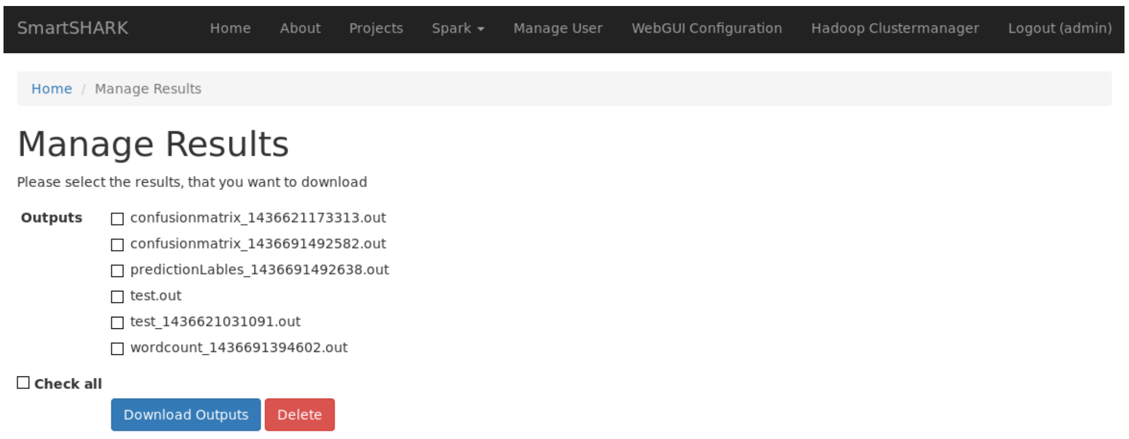
All Applications

Showing **1-10** of **69** items.

#	id	Name	Queue	State	Final Status	Progress	Start Time	Elapsed Time	Additional Information
1	application_1436439214173_0001	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 11:01:20	0 days, 00:00:20	👁
2	application_1436439214173_0002	org.apache.spark.examples.SparkPi	root.spark	KILLED	KILLED	100	09.07.2015 - 11:05:26	0 days, 00:16:21	👁
3	application_1436439214173_0003	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 11:24:09	0 days, 00:00:13	👁
4	application_1436439214173_0004	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 11:25:36	0 days, 00:00:12	👁
5	application_1436439214173_0005	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 11:26:00	0 days, 00:00:18	👁
6	application_1436439214173_0006	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 11:26:54	0 days, 00:00:17	👁
7	application_1436439214173_0007	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 11:27:41	0 days, 00:00:17	👁
8	application_1436439214173_0008	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 12:59:23	0 days, 00:00:56	👁
9	application_1436439214173_0009	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 13:02:20	0 days, 00:02:34	👁
10	application_1436439214173_0010	org.apache.spark.examples.SparkPi	root.spark	FINISHED	SUCCEEDED	100	09.07.2015 - 13:06:51	0 days, 00:02:15	👁

« 1 2 3 4 5 6 7 »

Figure 4.6.: GUI - Spark index. The figure shows, how different Spark jobs are displayed to the user.



Manage Results

Please select the results, that you want to download

Outputs
- ☐ confusionmatrix_1436621173313.out
- ☐ confusionmatrix_1436691492582.out
- ☐ predictionLables_1436691492638.out
- ☐ test.out
- ☐ test_1436621031091.out
- ☐ wordcount_1436691394602.out

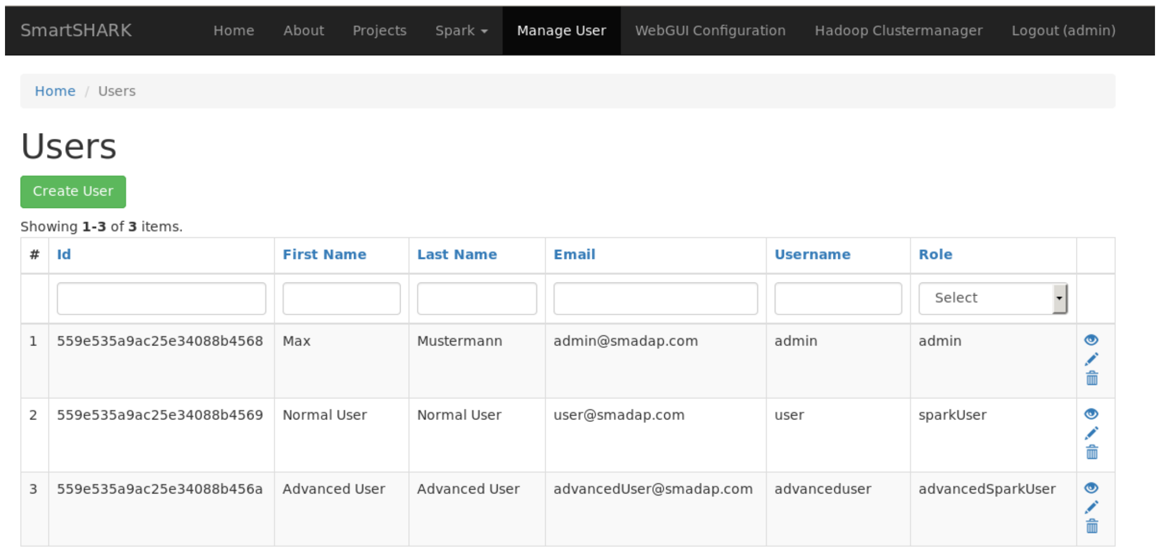
☐ Check all

Figure 4.7.: GUI - Spark results. The figure shows, how the results are displayed to the user. Each user has his own results, therefore only results saved in the corresponding user folder can be retrieved.

4.2.3. User Management

The platform implements Create, Read, Update, Delete (CRUD) user management functionalities. Figure 4.8 shows, how the available users are displayed. The users are displayed in a grid view, which illustrates the most relevant information. Furthermore, there exist the possibilities to view a user in detail (eye icon), edit a user (pencil icon) and delete a user (bin icon). Additionally, new users can be added via the create user button.

Each user of our platform has a role that is assigned via the web interface. The role specifies the permissions for both using the web interface, as well as accessing the MongoDB. There are three different roles, which are compared in Table 4.1. The three defined roles allow the separation between users who are allowed to trigger the mining of projects and adapt the configuration of the platform (Admin role), users that are allowed to create analytics that modify the MongoDB (AdvancedUser role) and users that can only create analytics that access the MongoDB without write privileges (User role). This can be useful if students should work with the platform to make sure that they cannot delete or overwrite data by mistake.



Users

Showing **1-3** of **3** items.

#	Id	First Name	Last Name	Email	Username	Role	
1	559e535a9ac25e34088b4568	Max	Mustermann	admin@smadap.com	admin	admin	👁 ✎ 🗑
2	559e535a9ac25e34088b4569	Normal User	Normal User	user@smadap.com	user	sparkUser	👁 ✎ 🗑
3	559e535a9ac25e34088b456a	Advanced User	Advanced User	advancedUser@smadap.com	advanceduser	advancedSparkUser	👁 ✎ 🗑

Figure 4.8.: GUI - User management. The figure shows, how users are displayed. Furthermore, it shows the different available actions: View a user in detail (eye icon), edit a user (pencil icon), delete a user (bin icon) and create a new user (button above the grid).

Role Name	Web Interface Permissions	MongoDB Permissions
Admin	All	All
AdvancedUser	All, except: Changing configuration files, mine, delete or backup projects, manage user, terminate Spark applications. Only fully mined projects are shown.	read, write, delete
User	All, except: Changing configuration files, mine, delete or backup projects, manage user, terminate Spark applications. Only fully mined projects are shown.	read

Table 4.1.: Comparison of user roles and their permissions in the web interface as well as in the MongoDB.

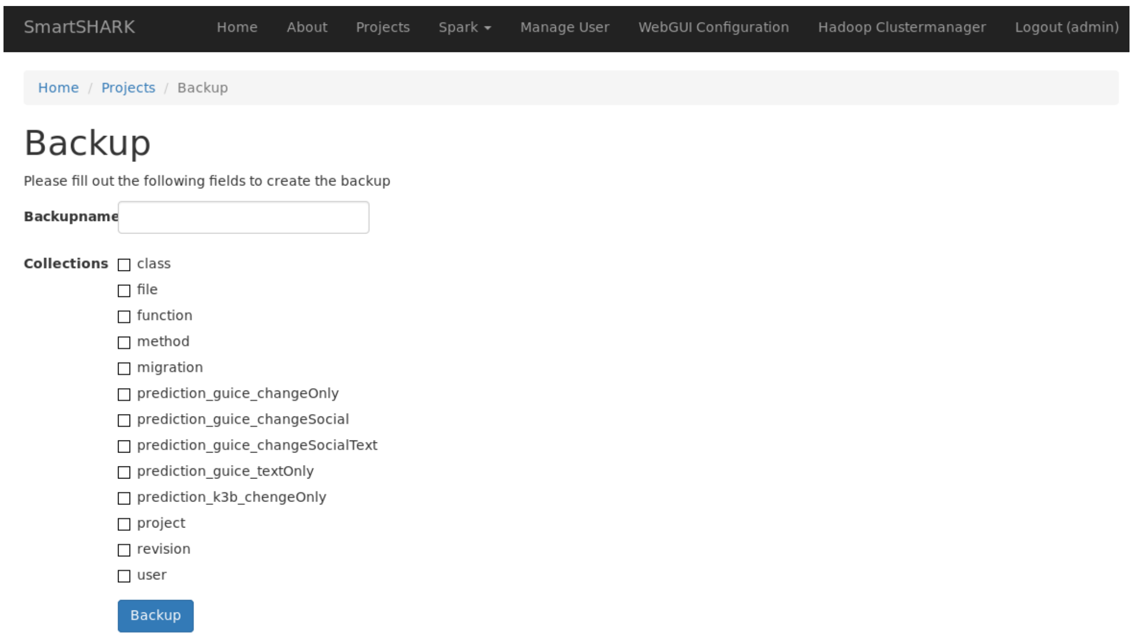
4.2.4. Additional Functionalities

Besides project management, project analysis and user management functionalities, there are other functions offered by the web interface, which are important for conducting research. First, it provides a backup functionality. The interface for this function is shown in Figure 4.9. The user needs to provide a backup name and must select, which collections should be backed up. A timestamp is added to the backup name and the resulting backup is saved as .tar.gz on the webserver. This feature is especially useful for the preparation of a publication, where a fixed data set is required which may not be changed afterwards.

Figure 4.10 illustrates, how backups can be managed. The user need to select the backup and can either download the backup or delete it via the corresponding buttons.

Furthermore, there exist the possibility to configure the connection to the MongoDB via the web interface. Figure 4.11 depicts, how the interface for this action looks like. It is important to note, that after changing the MongoDB connection, the administrator must rerun the MongoDB migration with the corresponding collection names, explained in Section 4.5. In the background, the configuration is saved as an XML file, because the DECENTMongoDBConverter (see Section 4.3.2) also needs this information.

Moreover, the web front-end was extended to show some analysis results for each project directly, instead of having the results only available as download. This was performed as part of the analytics example we provided to show how the platform can be modified to directly put results into operation and make them actionable. For further information, refer to Section 5.4.



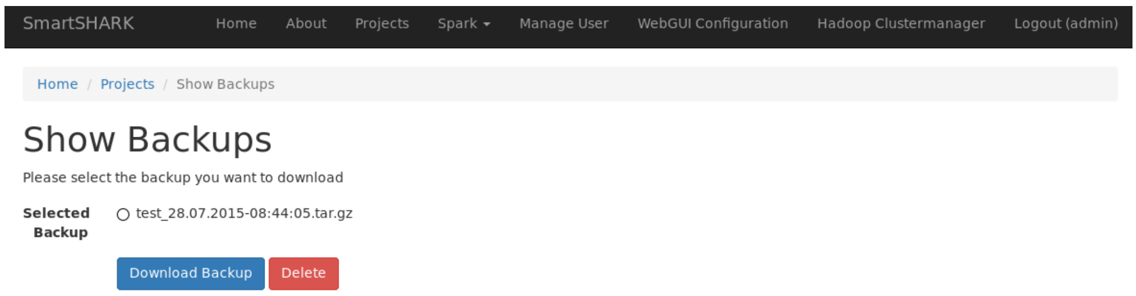
Backup

Please fill out the following fields to create the backup

Backupname

Collections
- ☐ class
- ☐ file
- ☐ function
- ☐ method
- ☐ migration
- ☐ prediction_guice_changeOnly
- ☐ prediction_guice_changeSocial
- ☐ prediction_guice_changeSocialText
- ☐ prediction_guice_textOnly
- ☐ prediction_k3b_chengeOnly
- ☐ project
- ☐ revision
- ☐ user

Figure 4.9.: GUI - Backup. The figure shows the backup functionality provided by the platform. The user can specify a backup name and the collections, which should be backed up.

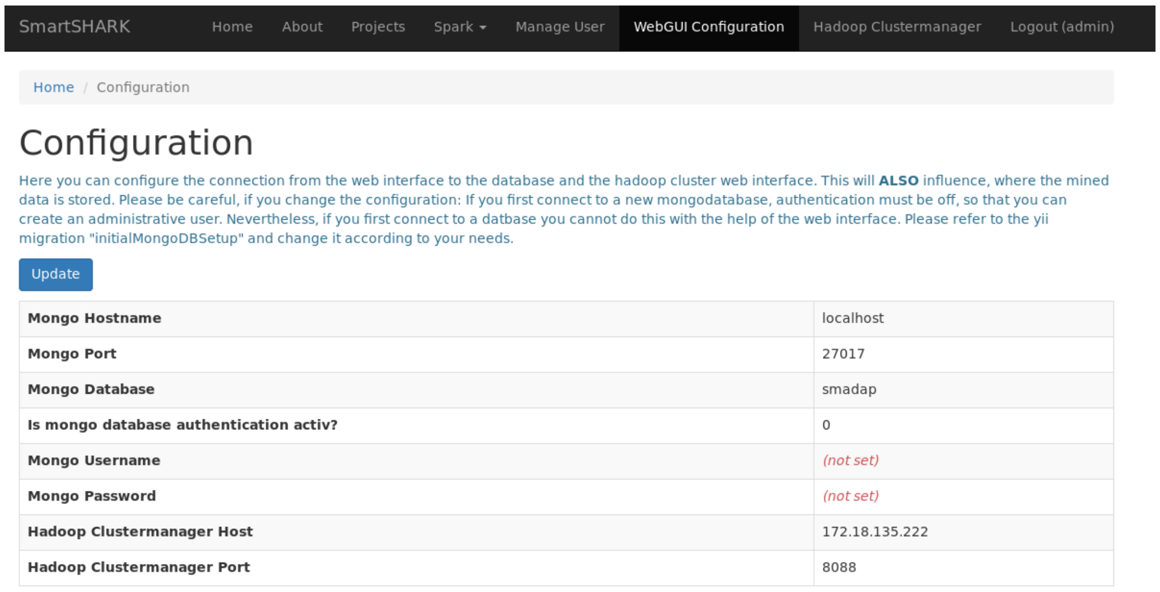


Show Backups

Please select the backup you want to download

Selected Backup ○ test_28.07.2015-08:44:05.tar.gz

Figure 4.10.: GUI - Backup management. The figure shows the backup management functionality provided by the platform.



Configuration

Here you can configure the connection from the web interface to the database and the hadoop cluster web interface. This will **ALSO** influence, where the mined data is stored. Please be careful, if you change the configuration: If you first connect to a new mongodatabase, authentication must be off, so that you can create an administrative user. Nevertheless, if you first connect to a datbase you cannot do this with the help of the web interface. Please refer to the yii migration "initialMongoDBSetup" and change it according to your needs.

Mongo Hostname	localhost
Mongo Port	27017
Mongo Database	smadap
Is mongo database authentication activ?	0
Mongo Username	*(not set)*
Mongo Password	*(not set)*
Hadoop Clustermanager Host	172.18.135.222
Hadoop Clustermanager Port	8088

Figure 4.11.: GUI - MongoDB configuration management. The figure shows the MongoDB configuration management functionality provided by the platform.

4.3. Project Mining

As described in Section 4.2, before a project can be mined it needs to be set up. The setup executes a shell script, which does the following steps.

1. **Create projects directory**, if it does not exist. In this directory, all project data is saved (e.g. intermediate EMF model instances).
2. **Create needed subdirectory** for the project, if these do not exist (e.g. log directory).
3. **Setup CVSAanaly database**: CVSAanaly saves the results into a MySQL database. This database needs to be set up beforehand.
4. **Setup DUDE database**: DuDe saves the results into a MySQL database. This database needs to be set up beforehand.
5. **Setup tool configrations**: During the mining of the project, several tools for converting intermediate results are used. Each of these tools has its own configuration file. These files need to be adapted to the current project (e.g., set project name in configuration file) and copied to the project folder. This includes the RT, MX and Mel configuration files. For further information about these tools, refer to Section 4.3.1.

After the setup is done, the project can be mined. If the user decides to mine a project, another shell script is executed. First, a DECENT model instance is created, which is then transformed into another EMF model instance (called DECENTMongo model) and after that written into the MongoDB. The DECENT creation step is explained in detail in Section 4.3.1. The conversion of the DECENT model and the saving of the results in the MongoDB is illustrated in Section 4.3.2.

4.3.1. DECENT Creation

As it is described above, one fundamental step in the mining process is the creation of the DECENT model file. DECENT is introduced in Section 2.1.

After the project was set up, the user can start the mining process. This triggers the execution of a bash script on the webserver. Important to note is, that we must distinguish between a project, which was mined before and a new project. If the project was mined before, we do not need to copy the different configuration files described above and the whole setup process does not need to run completely.

Listing 4.1 shows an excerpt of the used bash script. It illustrates how the different phases in the mining process are executed: The first step is to write the current phase, which is executed, to the log file (see lines 2 and 13). This ensures, that we can track back problems, if they occur. After that, the corresponding phase is performed. Hence, another bash script is executed, which contains commands specific for this phase (see lines 3 and 14). Afterwards, the main script waits, till the phase script is completed (see lines 4 and 15). A plain sanity check is performed, which checks either, if the programs return value is not zero (see lines 5-8) or if the EMF model file, which should be created, was created (see lines 16-20). If the sanity check fails, an error string is written to the state file (see lines 6 and 18), which contains the different states completed by the project. With the help of this file, the progress of the project can be shown in the GUI (see Section 4.2.1). Afterwards, a timestamp and a message is written to the log file (see lines 9 and 21) and a phase completed string is written to the state file (see lines 10 and 22) to make clear, that this phase was successfully executed.

```
1  # Run cvsanaly
2  echo "Run CVSAnaly..." >> $LOGPATH
3  $DEPLOYMENT/scripts/cvsautomate $DATABASE $DEPLOYMENT/projects/$PROJECTNAME/
       git >> $LOGPATH 2>&1
4  wait
5  if [[ $? -ne 0 ]] ; then
6          echo "::!error!::" >> $STATEPATH
7          exit 1
8  fi
9  echo "$(date +%d.%m.%Y-%T) CVSAnaly completed" >> $LOGPATH
10 echo "$(date +%d.%m.%Y-%T) ::!cvsanalycomplete!::" >> $STATEPATH

12 # Run MG
13 echo "Run RT-MG..." >> $LOGPATH
14 $DEPLOYMENT/scripts/run-rt-changed.sh $PROJECTNAME >> $LOGPATH 2>&1
15 wait
16 if [ ! -e "$OUTPUTPATH/model.mg" ]
17 then
18         echo  "::!error!::" >> $STATEPATH
```

```
        exit 1
fi
echo "$(date +%d.%m.%Y-%T) Run MG completed" >> $LOGPATH
echo "$(date +%d.%m.%Y-%T) ::!runmgcomplete!::" >> $STATEPATH
```

Listing 4.1: Excerpt of the project mining bash script.

It is important to note, that we create a separate log file for the InFamix step, because the InFamix log file can get huge, depending on the size of the project and the number of commits. Furthermore, we developed a DECENT importer, which gets a previously created DECENT file as input and executes only the last two steps. In Table 4.2, we shortly summarize the different steps and their created output. DECENT provides several programs, which are used for converting the extracted facts and creating a DECENT model instance. The resource tools (RT) program is used to convert different facts assets (e.g. the MySQL database created by CVSAnaly) into EMF model instances. The MX tools provided by DECENT is a newer version of the FX tools, described in Section 2.1. The MEL tools are used for the integration step, where the different facts model instances are combined into one DECENT model instance.

Phase	**Description**	**Input**	**Output**
Pull project	Executes git clone on project repository location	Repository location	Repository data
CVSAnaly	Executes CVSAnaly (see Section 2.1) on the before cloned project data	Cloned project data	MySQL database, with project history data (e.g. different revisions and actions)
RT-MG	Uses the resource tools provided by DECENT to create an intermediate EMF model	CVSAnaly MySQL database	MG model instance

MX-Famix	Uses the mx tools provided by DECENT to run InFamix on each revision. Checks out each revision to run InFamix on it	Revisions	MSE files[5], for each revision
RT-Famix	Uses the resource tools provided by DECENT to create an intermediate EMF model	MSE InFamix files	FAMIX model instance
RT-DAG	Uses the resource tools provided by DECENT to create an intermediate EMF model in two steps (generation of DAGX, translation to DAG)	Cloned project data, DAGX	DAG, DAGX
MX-DuDe	Uses the mx tools provided by DECENT to run DuDe on each revision	Revisions	Dude files
RT-DuDe	Uses the resource tools provided by DECENT to create an intermediate EMF model	Dude files	DUDE model instance
Mel-Workflow	Uses the mel tools provided by DECENT to link and process all existent intermediate model files to create a DECENT file	MG, FAMIX, DAG, DAGX and DUDE models	DECENT, CFA
Decent to DECENT-Mongo	Uses the DECENTMongoDB-Converter (see Section 4.3.2) to create a DECENTMongo model file out of the DECENT model file	DECENT model file	DECENTMongo model file

[5]MSE is a specific file format used by Moose. See: `http://www.moosetechnology.org/`.

DECENT-Mongo to MongoDB	Uses the DECENTMongo-DBConverter to write the DECENTMongo model file, after some processing, in the MongoDB	DECENTMongo file	MongoDB documents

Table 4.2.: Different phases in the project mining process, with a short description and their in- and output.

4.3.2. DECENTMongoDBConverter

We developed a tool called "DECENTMongoDBConverter", which is used to write the contents of a DECENT file (see Section 2.1) into a MongoDB (see Section 2.4). As intermediate result, we create a DECENTMongo model instance, which is based on EMF. We decided for this approach, because creating an EMF model via EOL is straight forward. EOL provides a querying language and therefore, we can query the data that we want to save in our DECENTMongo model file easily. Therefore, we designed our DECENTMongo model similar to our MongoDB design (see Section 3.3.2). The design of our DECENTMongo model is depicted in Figure 4.12.

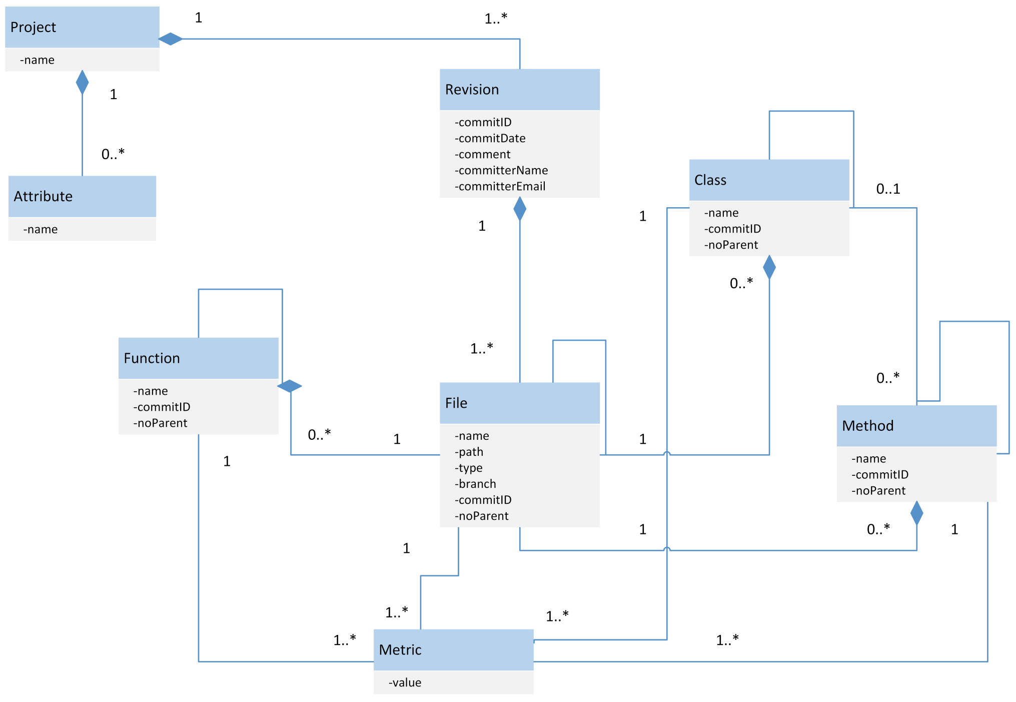


Figure 4.12.: Design of the DECENTMongo model.

The second step is to write the contents of the DECENTMongo model into the MongoDB. For this, we extended our DECENTMongoDBConverter: The DECENTMongo model is read using EMF and the contents are converted into Java objects. These Java objects are annotated with the Morphia framework (see Section 2.4). This enables an easy writing of the contents to the MongoDB. We have chosen Morphia for this task, because it has two advantages: First, we can use Morphia to query data from the MongoDB. The return value is a Java object, which possesses the values of the retrieved document in the MongoDB. Hence, we can work with these Java object, like normal objects but in the background these objects are connected to the MongoDB. Secondly, the handling of the connection to the MongoDB is easy via Morphia.

4.4. Utilize the Data with Apache Spark

We followed our design presented in Section 3.4. The implementation is depicted in Figure 4.13. The user writes the analysis program, based on Apache Spark, which then can be uploaded to the webserver, using an upload form (see Section 4.2.2). The front-end allows to add Spark arguments as well as program arguments. This allows the creation of parameterizable analytic jobs, e.g., to define the name of the project to be analysed using a program argument. In the background, Apache Sparks "spark-submit" script is invoked, with the uploaded program and given parameters. Hence, an Apache Spark job is submitted to the Apache Hadoop cluster. After that, the JAR or Python file is distributed among the nodes in the cluster and processed in parallel. Apache Spark uses RDDs for this parallelization step, which are explained in Section 2.3. Information about all jobs, both running, as well as completed, are retrieved via the REST API of Apache Hadoop. Hence, we implemented a connection to the Apache Hadoop REST API to display the retrieved information in a grid (see Section 4.2.2). The REST API does not provide all available information about an application. But it is possible to directly access the Apache Hadoop web interface, if the displayed information is not enough. Furthermore, we implemented the possibility to terminate a running job directly via the web interface.

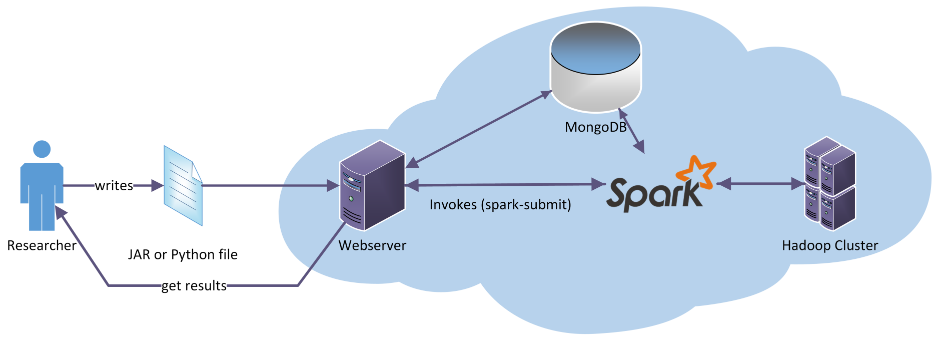


Figure 4.13.: Implementation of the analysis based on Apache Spark and Apache Hadoop. The MongoDB as well as Apache Spark are running on the webserver.

The uploaded files are temporarily saved in the Yii application structure. We created a cronjob, which is executed at midnight every day, that deletes this upload folder. Results

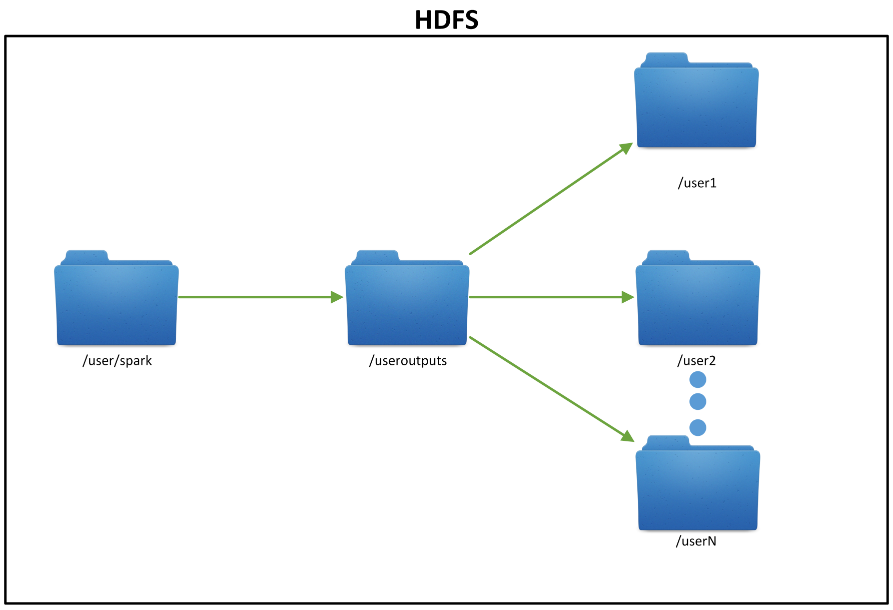


Figure 4.14.: HDFS folder structure for the user "spark".

of the Spark jobs can be saved in the MongoDB (if the user has sufficient rights, see Section 4.2.3) or in the HDFS. Every user of the platform has her own directory, where data can be saved. The structure of this is depicted in Figure 4.14. In the system, there exist an Apache Spark user called "spark". This user is used for all the actions performed by the webserver. Furthermore, this user has its own HDFS folder. A sub directory of his HDFS home folder is the folder called "useroutputs". There, every user that is created via the web interface (and has submitted a Spark job) has its own directory. The user has full control over his directory. Hence, the user can delete, download and save new files to it (see Section 4.2.2). If the user wants to download more than one file, the files are automatically packed into a tar.gz and a timestamp is added. Nevertheless, if the user saves his outputs (e.g. a text file) to the wrong location, there is no way to download or delete this file by this user.

4.5. Cloud Deployment

As it is explained in Section 3.5, we want to achieve, that the whole infrastructure can be set up with minimal user interaction in a cloud environment. To achieve this, we use the tools Vagrant (see Section 2.6.1) for the creation of the VMs in the cloud and Ansible (see Section 2.6.2) for the VMs configuration. In order to be able to deploy our platform, the cloud environment must support Vagrant and Ansible. Usually, a cloud environment provides dedicated configuration servers, to which Vagrant and Ansible scripts can be uploaded. In our case, this is a dedicated server with a connection to the OpenStack installation of our cloud. To run the deployment from the dedicated server, only two bash commands must be called: *vagrant up –no-provision* for the creation of the VMs and *vagrant provision* for their configuration.

Figure 4.15 shows the infrastructure after the deployment. This figure does not show the communication flow in the Hadoop Cluster. As explained in Section 3.5, the four different parts of our platform should be separated. However, we were not able to deploy our platform this way, as we only have limited resources available. Nevertheless, our deployment scripts can be easily adapted to achieve the separation of the platform parts. A more efficient design is presented in Chapter 7.

Note, that if we describing Apache Hadoop *services* in this section, we will write it in camel case (e.g. NameNode). If we mention our *hosts* that we have created, we write it in lower case (e.g. namenode).

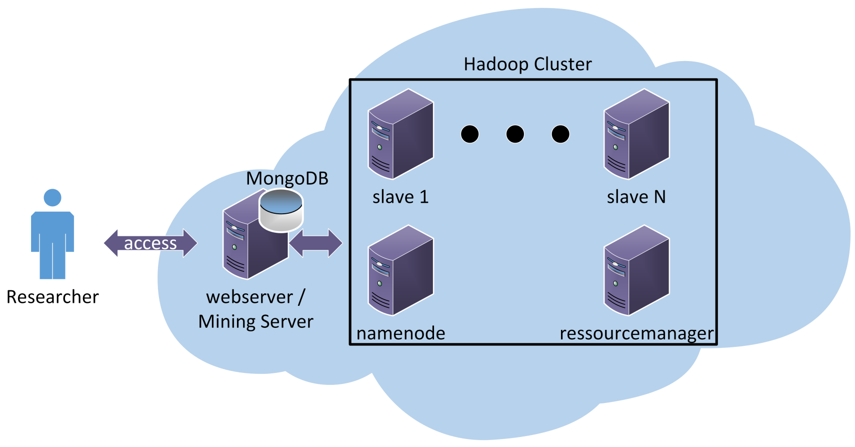


Figure 4.15.: Infrastructure after Vagrant and Ansible are called. The communication flow in the cluster itself is not depicted.

4.5.1. Creation of VMs

We use Vagrant for the creation and provisioning of the required VMs for running our platform in a given cloud environment. We adapted the settings used in [48]. The user can define the amount of slaves, which should be present in the Hadoop cluster. Furthermore, the so-called flavours can be defined. Listing 4.2 shows how the slaves are instantiated using Vagrant. Fore each iteration, a VM is defined via "config.vm.define". Furthermore, we directly assign a hostname to it.

```
SLAVES_COUNT.times do |i|
  config.vm.define "slave#{i+1}" do |slave|
    slave.vm.hostname = "slave#{i+1}"
    end
end
```

Listing 4.2: Slaves definition in the used Vagrantfile

Additionally to the slaves, we need to create a namenode and a resourcemanager for the Hadoop cluster. Furthermore, the webserver needs to be created and provisioned. It is to note, that at our current setup the webserver needs to have a huge amount of RAM, as the mining process is executed on it and the created EMF models need to fit completely into the main memory.

Once the VMs are created, they must be provisioned, i.e., instantiated with an operating system. Vagrant allows the selection of predefined images of operating systems which can be downloaded for this purpose. Once the provisioning is also performed, we have the required VMs up and running with an operating system, but no other software installed yet, as this is not supported by Vagrant.

Therefore, Vagrant calls Ansible to configure the different nodes. Listing 4.3 shows how Vagrant and Ansible can interact with each other. This listing illustrates, how the resource-manager is prepared for the configuration via Ansible: First, we need to configure Ansible (see lines 5-8). Important to note is, that we need to specify the playbook, which should be used for the configuration (see line 7). Furthermore, different hosts are grouped together (see lines 11-16). Hence, they can be referenced with the given names in the playbook (e.g., "Slaves" references all created slave VMs). The full source code is given in appendix A.

```
1  config.vm.define "resourcemanager" do |resourcemanager|
2      resourcemanager.vm.hostname = "resourcemanager"

4      resourcemanager.vm.provision :ansible do |ansible|
5          ansible.verbose = "v"
6          ansible.sudo = true
7          ansible.playbook = "deployment/site.yml"
8          ansible.limit = "all"

10         slaves = (1..SLAVES_COUNT).to_a.map {|id| "slave#{id}"}
11         ansible.groups = {
12             "NameNode" => ["namenode"],
13             "ResourceManager" => ["resourcemanager"],
14             "Slaves" => slaves,
15             "Webserver" => ["webserver"]
16         }
17     end
18 end
```

Listing 4.3: Vagrant-Ansible interaction

4.5.2. VM Configuration

As stated above, Ansible is used for the configuration. Ansible can be called using Vagrant and allows the definition of so-called playbooks, i.e., scripts that define which software should be installed in a VM and how the software should be configured. It is possible to assign different playbooks (or tasks in playbooks) to different created groups. Hence, not every playbook is executed by every group. Table 4.3 gives an overview of the different playbooks and their purpose.

Playbook	**Purpose**
common	Installation of required base software
hadoop_common	Hadoop download and system preparation
configuration	Hadoop configuration
format_hdfs	HDFS formatting
services	Hadoop services start/restart
setup	Creates the webserver

Table 4.3.: Ansible playbooks for the cluster deployment. Based on [48].

The configuration of our platform requires six different playbooks. Each playbook serves a different role for the configuration of the platforms infrastructure. The playbooks are the following:

- **common**: Installs required base software, which is needed on every node in the cluster (e.g. Java, Network File System (NFS)).
- **hadoop_common**: Downloads a specified Hadoop distribution (for our experiments we used Apache Hadoop 2.6) and creates the Hadoop user on every node. This is also required on the webserver, because Apache Spark needs access to Hadoop.
- **configuration**: Configures Apache Hadoop on every node. It also makes the HDFS available via a Portable Operating System Interface (POSIX) compatible file system using a NFS wrapper.
- **format_hdfs**: Is executed on the namenode. Formats the HDFS.

Service	**Hosts**
NameNode	namenode
ResourceManager (YARN)	resourcemanager
JobHistoryServer	resourcemanager
Portmap (required for NFS wrapper)	namenode
NFS Wrapper	namenode
NodeManager	All slaves
DataNode	All nodes
Apache2	webserver
MongoDB	webserver
MySQL	webserver

Table 4.4.: Services running on different node types. Based on [48].

- **services**: Is executed on each node. Different needed services, like the NodeManager on the slaves or YARN on the resourcemanager are started. Table 4.4 gives an overview of the different services, which run on the different nodes.
- **setup**: Only the webserver executes this playbook. It contains tasks, like the installation of an Apache2, creation of the site configuration for the web interface, installation of needed software (e.g. GIT, PHP5, CURL, MongoDB, MySQL, Apache Spark), installation of plug-ins (e.g. PHP-MongoDB extension) and execution of a Yii migration. The purpose of the Yii migration is the instantiation of the MongoDB user collection with default values. Hence, after the instantiation it is possible to log in into the web interface with the specified default users.

5. Case Study

In this chapter, we examine the practicability of our platform. We use our platform to mine and analyse different software projects. First, we evaluate the mining part of the platform by cloning multiple projects and collecting their data. We developed a sample analysis in which we show how defect prediction models can be created for the previously mined projects. Moreover, we adapted the platforms front-end to directly show the results of our sample analysis in order to allow quick feedback to researchers and developers interests in such analysis and to provide a major step towards making the analysis actionable.

First, the test environment is described in Section 5.1. Then we introduce the projects we have chosen for our case studies in Section 5.2. In Section 5.3 we evaluate the mining process of these projects. In the last section (see Section 5.4), we conduct defect prediction research on the mined projects and illustrate, how the results are presented via the web interface.

5.1. Environment

Currently, we are running our platform in a small cloud that we locally administrate within our research group[1]. The cloud is based on OpenStack. As it is explained in Section 4.5, we need to specify the flavour for the machines, as well as how many slaves our Apache Hadoop cluster should include. Table 5.1 shows the created machines and their specifications.

The webserver needs to have a huge amount of RAM, because it also performs the mining and the EMF models created during the mining must fit completely in the main mem-

[1]See: `http://www.swe.informatik.uni-goettingen.de/`.

Hostname	Virtual CPUs	Memory	Storage	Architecture
namenode	2	4GB	40GB	64bit
resourcemanager	2	4GB	40GB	64bit
slave1	2	4GB	40GB	64bit
slave2	2	4GB	40GB	64bit
webserver	8	16GB	160GB	64bit

Table 5.1.: Infrastructure of our deployed case study environment.

Software	Version
Apache Hadoop	2.6.0
Ansible	1.9.2
Vagrant	1.7.2
Java	OpenJDK 64-bit 1.7.0_79
Linux	Ubuntu Server 14.04
MySQL	5.5.43
MongoDB	3.0.4
Apache2	2.4.7
PHP	5.5.9
Apache Spark	1.4.0
Yii	2.0.5

Table 5.2.: Software versions of our deployed case study environment.

ory. Furthermore, Table 5.2 lists all software that we have used, which are directly relevant for our platform. On which node the software is installed, is explained in Section 4.5.

5.2. Project Descriptions

To evaluate if the mining of projects works as desired, we selected some projects from GitHub[2]. We did not follow any specific methodology for the selection of projects, but selected 20 projects randomly by using the "explore" function of GitHub. Furthermore, we checked, if their programming language is Java or C/C++, as this is the only requirement

[2]See: `https://github.com/`.

of our platform. Table 5.3 lists the mined projects, including the programming language, the number of commits, the number of files in the repository, the size of the repository, and a very brief project description. The number of files and the size are reported for the latest revision of the repository because we assume, that the last revision is the biggest revision with the most files. But this assumption might be wrong for some projects. Nevertheless, we only want to illustrate the approximate size of the used projects to make it possible to set the size of the project in relation to the duration of the mining and analysis of it evaluated in Section 5.3 and 5.4.

Project	**Lang.**	**#Commits**	**Size**	**#Files**	**Description**
cursynth [84]	C++	219	3 MB	185	MIDI enabled, subtractive synthesizer, for the terminal.
cxxnet [28]	C++	852	4 MB	173	Concise, distributed deep learning framework.
elasticsearch-hadoop [35]	Java	1243	9 MB	576	Elasticsearch real-time search and analytics natively integrated with Hadoop.
fatal [36]	C++	401	3 MB	141	Library for fast prototyping of software in C++.
guice [46]	Java	1442	89 MB	713	Dependency injection framework for Java.
HackerNews [64]	Java	12	1 MB	78	Pulls top stories from the HackerNews API for Android.
libxcam [1]	C++	250	3 MB	242	Project, with the aim to extend camera features.
libyami [2]	C	487	4 MB	248	Library for media solutions on Linux.
Minesweeper [65]	Java	65	7 MB	207	Minesweeper game for Android.

mxnet [29]	C++	236	1 MB	124	Combines ideas from projects for machine learning.
oclint [78]	C++	733	3 MB	349	Static source code analysis tool.
ohmu [47]	C++	226	3 MB	185	Compiler intermediate language for static analysis.
openage [81]	C++	1761	8 MB	560	Project, which clones Age of Empires II.
oryx [22]	Java	372	48 MB	537	A real-time large-scale machine learning infrastructure.
osquery [37]	C++	2208	9 MB	555	Operating system instrumentation framework.
passivedns [40]	C	220	1 MB	50	Network sniffer, which logs DNS server replies.
SMSSync [85]	Java	1395	40 MB	557	SMS gateway for Android.
swift [38]	Java	496	8 MB	427	Annotation-based Java library for Thrift.
wds [3]	C++	238	3 MB	193	Libraries for building Miracast/WiDi-enabled applications.
xgboost [30]	C++	1847	8 MB	373	Large-scale, distributed gradient boosting library.

Table 5.3.: Information about the mined projects.

5.3. Mining the Projects

The mining of the projects was very time intensive. For each of the selected projects, we triggered the mining via the web front-end of our platform, as it is described in Section

4.2.1. We measured how long the different steps of the mining process took and recorded if we encountered problems. Figure 5.1 shows how long each phase took to complete for each project. The differences in the overall runtime between the projects are a result of the differences in project size, number of files, and number of commits. Even though the mining is only runnning on a single core of the webserver, the largest project took about one and a half days to mine. Most time is consumed by the collection of the metric data with MX-Famix and the DECENTMongoDBConverter. There are two outliers, where the runtime is consumed differently. The first is guice, where the addition of the diffs in the RT-MG step consumes most of the runtime. The second outlier is the oryx project, for which the analysis of the repository itself consumed the most time. When we investigated this, we determined that CVSAnaly stopped working for a while, but then simply resumed as if nothing happened. Upon further investigation we determined that this was an issue with CVSAnaly due to a race condition in the implementation.

For Mahout[3], the mining simply stopped due to a failure of InFamix. InFamix did not throw any errors, it just stopped working. Therefore, the process did not finish. We were not able to track down the problem. But besides this incident, there were no other failures in mining for the projects that we have tested. But, during our tests we faced some limitations, which are described in detail in Chapter 7.

[3]See: `https://github.com/apache/mahout`.

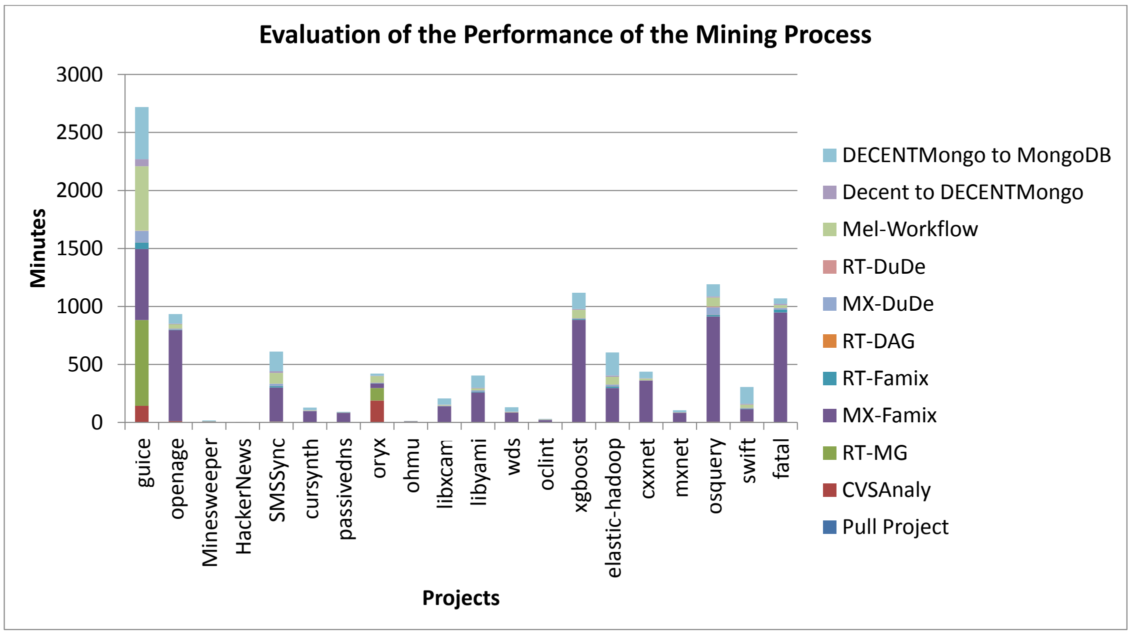


Figure 5.1.: Evaluation of the performance of the mining process. The mining was performed on the webserver. One core and 12 GB RAM were used at peak for this process.

5.4. Application Example: Defect Prediction

Defect prediction is an important research topic nowadays (see [53, 93]). It is used to identify defect-prone entities (e.g. source code files) in advance. Typically, defect prediction models use a training set, which contains various measures (e.g. code metrics) of available entities and the history of the entities defect proneness. The prediction model is fit to the training set. Afterwards, the model is used to predict defects of entities in the future. This preselection of entities, which most likely contain a bug, is done, as budget, people and time, are often limited in development organizations and therefore only some of the code can be tested before releasing it [76].

Therefore, we created different classifier with different metrics for the mined projects to predict bugs in the mined data. We used our platform to execute the analysis. The conducted defect prediction research is not designed to have the highest performance possible or implement a very complex defect prediction scheme. Instead, we want to demonstrate how the different data offered by our platform can be used and combined for an analytic goal. We use defect prediction to figure out, which files of a commit are most likely to contain a bug. It is important to note, that we only look at file artifacts. Therefore, we do not use classes, methods or functions for our application example.

First, we explain how we have chosen our training and test data (see Section 5.4.1). In Section 5.4.2, we describe our classification models in detail. Furthermore, we describe one analysis program for building one of these classifier as an example in detail in Section 5.4.3. At last, the results of our defect prediction models are presented in Section 5.4.4.

5.4.1. Training and Test Data

We are using a within-project defect prediction approach [76]. Therefore, we built our defect prediction model on the past of a project and evaluate it on later revisions (of the same project). Hence, data from other projects are not used during the training of a prediction model. To select the training and test data within a project, we took a pattern from [83]: We leave a short gap in the beginning, i.e., we are excluding the first commits of a project, because the change patterns may not be stable in the beginning [52, 61]. We take the next commits for training, then we leave a gap, and then we use the next period of the project to evaluate the prediction model. Moreover, we do not consider all commits until the end of the project, but leave another gap here, because this data is fault-prone, as there was less time for the identification of bugs. We use the information from the origin analysis [44] of DECENT, which is based on the bug labels assigned by CVSAnaly's *BugFixMessage* extension. This information is already contained in the data through our mining process.

This does not apply to all projects, as the lifetime of some projects that we have mined is less than three years. Therefore, we did not exclude the first commits for these projects. Furthermore, we cut out the last half of commits for those projects (instead of three years), as we do not have any data about bug fixing times available. Additionally, we exclude the project "HackerNews", as it only has 12 commits. Hence, there is not enough data to perform defect prediction analysis.

5.4.2. Classification Models

We created four different classification models with the Mllib of Spark, which use different metrics provided by our platform. Training defect prediction classifier based on software metrics is a commonly used method in research [19].

- **CL1**: The first classification model is based only on change and static source code metrics. Using these metrics as input, we trained a binary logistc regression model for the classification.
- **CL2**: The second classification model is based on change, static source code and social metrics. Using these metrics as input, we trained a random forest for the classification. In the literature, we know only one example where all three kinds of metrics were combined, however, this was not in the context of defect prediction, but vulnerability prediction [82].
- **CL3**: The third classification model is based on only on the *diff* characteristic for text classification. *Diff* is calculated by the differences between the last state and the actual state of the artifact. Hence, all changed lines (removed or added) are present in the *diff*. We calculate the tf-idf [77] for the different *diffs* and use this as classification metric. Based on this metric, we trained a naive bayed classifier for the classification into defect and non-defect prone files.

- **CL4**: The fourth classification model is a combination of CL2 and CL3. It was created to demonstrate, that our platform allows the combined analysis based on software metrics and text analysis. In the third model we internally train three different classifier: Logistic regression and random forest, which are both trained with change and social metrics and a naïve bayes classifier, which is trained with the tf-idf value described above. Then we use an ensemble learning method called bagging [88] to determine the overall classification. All three classifier classify an instance. Then, we use the majority vote of those three internal models to determine the overall classification of the instance.

We have chosen logistic regression, random forest and naïve bayes as classifier, because they were available in Sparks Mllib and they are often used in the field of defect prediction. The results of our classification are written back to the MongoDB in a new collection (for further information refer to Section 5.4.3). Through storing the analysis results in the database back-end, we enrich the database with more information that was not gained through the mining directly, but rather with advanced software analytics based on our platform itself. Therefore, our platform can feed itself with newly created data. Furthermore, the newly created collections are used to operationalize our software analytics approach: Our web interface is able to visualize prediction results (for further information refer to Section 5.4.4).

5.4.3. Description of a used Apache Spark Program

As an example, we explain our Apache Spark analysis program for creating classifier CL1. The full source code can be found in Appendix B. The first step is the creation of a Spark context (see lines 102, 103). This context is needed, to specify the connection to Apache Spark. The next step, as we are using deep-spark introduced in Section 2.4, is to create a deep mongo config (see lines 105-117). Together with the Spark context, we can specify what data should be queried and which data of a document should be returned. We created our analysis program in a way, that it needs the following parameter to work:

1. **project name**: The name of the project to analyse, like it was specified in the web GUI.

2. **training begin commit id**: Specifies, on which commit id the training of the classifier begins.
3. **training end commit id**: Specifies, on which commit id the training of the classifier ends.
4. **classification start commit id**: Specifies, on which commit it the classification of the classifier starts.
5. **classification end commit id**: Specifies, on which commit it the classification of the classifier ends.
6. **classifier name**: Specifies the name of classifier/analysis project. This is important for displaying the information in the web interface. Further details are given below.

The next step is the sampling of the training data (see lines 119-120). This is done, because most of the data is highly unbalanced. Hence, there are a lot more non bug-prone instances than bug-prone instances. If we do not sample our training data, the classifier will be strongly influenced by this imbalance problem. After that, we need to preprocess the data. Therefore, we need to create labeledPoints, which is a class used by Apache Spark to train and evaluate classifier (see lines 131-189). We use the map function, which gets an input (in this case Cells) and creates an output via a specified function. As next step, we create the classifier (in this case a logistic regression classifier, see line 195) and get the classification data (see lines 197 - 219), preprocess it in the same way (see lines 228 - 278) and evaluate our classifier (see lines 281 - 316).

Listing 5.1 illustrates, how we create a RDD of Cells via the map function, which contains the prediction (see lines 10 and 14) and the corresponding file id (see lines 8 and 13). We create this RDD, because we want to read out these information later on via the webserver. Furthermore, later in the program, we also calculate True Negatives (TN), True Positives (TP), False Negatives (FN), and False Positives (FP) and save them in this RDD as well. As last step, we save this RDD in the MongoDB.

```
1 // Create prediction set for mongodb
2 JavaRDD<Cells> idandlabel = classificationData.map(
3   new Function<Map<String, LabeledPoint>, Cells>() {

5     public Cells call(Map<String, LabeledPoint> map) {
      // Get labeledpoint
      String id = (String) map.keySet().toArray()[0];
      LabeledPoint point = map.get(id);
      Double prediction = model.predict(point.features());

      Cells result = new Cells();
      result.add(Cell.create("fileId", id));
      result.add(Cell.create("prediction",getLabelForDouble(prediction)));

      return result;
    }
  }
);
```

Listing 5.1: Computation of raw scores on test set and creation of a RDD which is to be saved in the MongoDB.

Important to note is, that the collection that is created follows a specified naming scheme. The scheme is "prediction_<projectname>_<name>". <projectname> is substituted by the project name given as input for this program and <name> is substituted by the classifier name given as input. Furthermore, it is important to note, that if we run the program with the same input twice, it will delete the existing collection first and create a new one with its results.

5.4.4. Results

Figures 5.2 - 5.4 depict the seconds till the classifier were build and evaluated by Spark. Figure 5.2 puts this into relation to the number of commits of the projects. Whereas Figure 5.3 puts the time till the classifier were created and evaluated in relation to the project size and Figure 5.4 uses the number of files as factor. These figures illustrate, that the time till the classifier were created and evaluated does not raise with bigger projects (bigger in terms of number of commits, files or size of the project), but we can identify a nearly linear trend. The peak for this task is at about 2 minutes for the most complex classifier we have created.

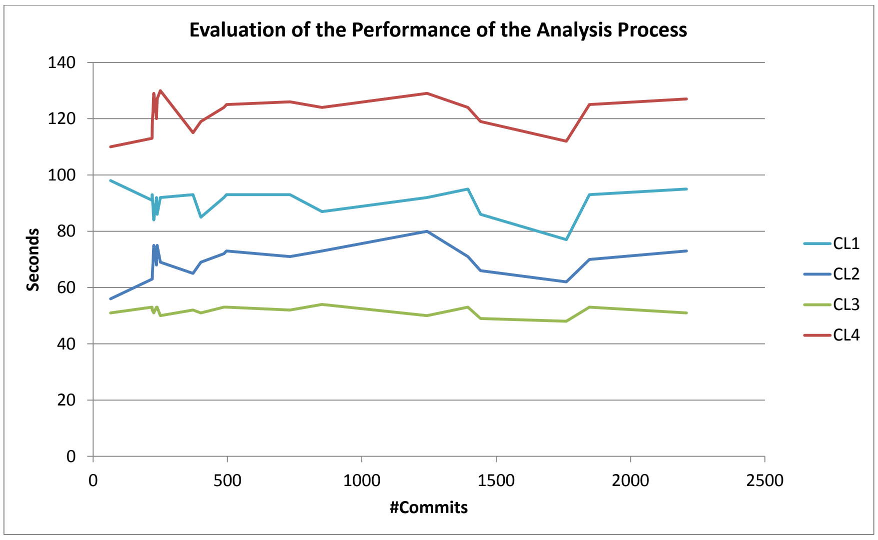


Figure 5.2.: Evaluation of the performance of the analysis process using Apache Spark for the chosen projects. The figure depicts the seconds till the classifier were build and evaluated by Apache Spark. This time is put into relation to the number of commits.

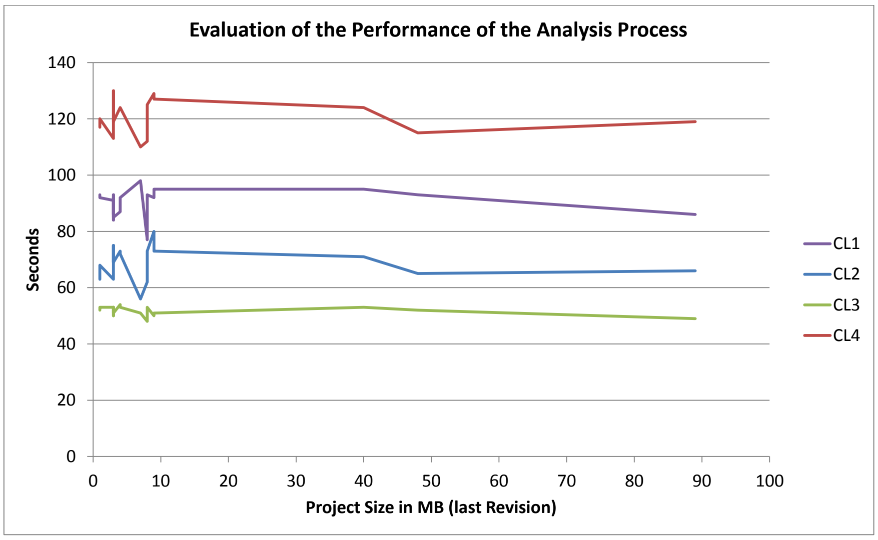


Figure 5.3.: Evaluation of the performance of the analysis process using Apache Spark for the chosen projects. The figure depicts the seconds till the classifier were build and evaluated by Apache Spark. This Time is put into relation to the projects size.

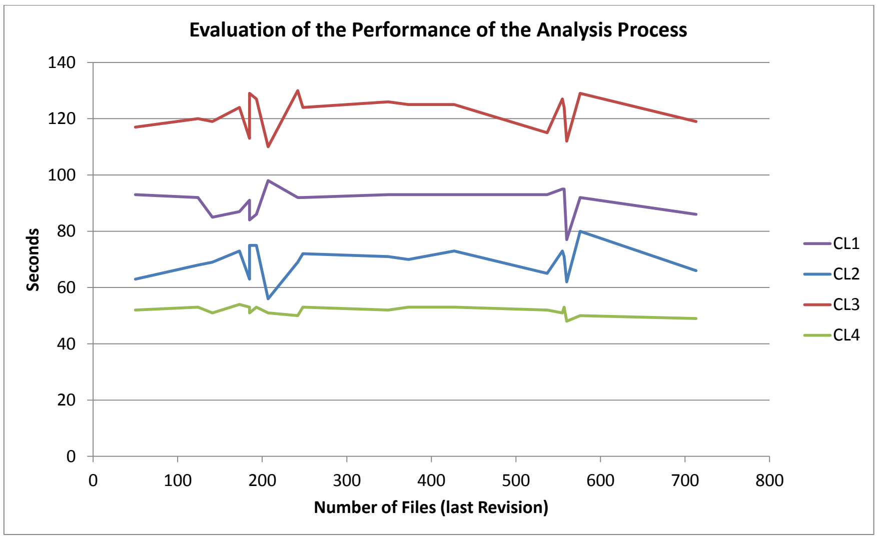
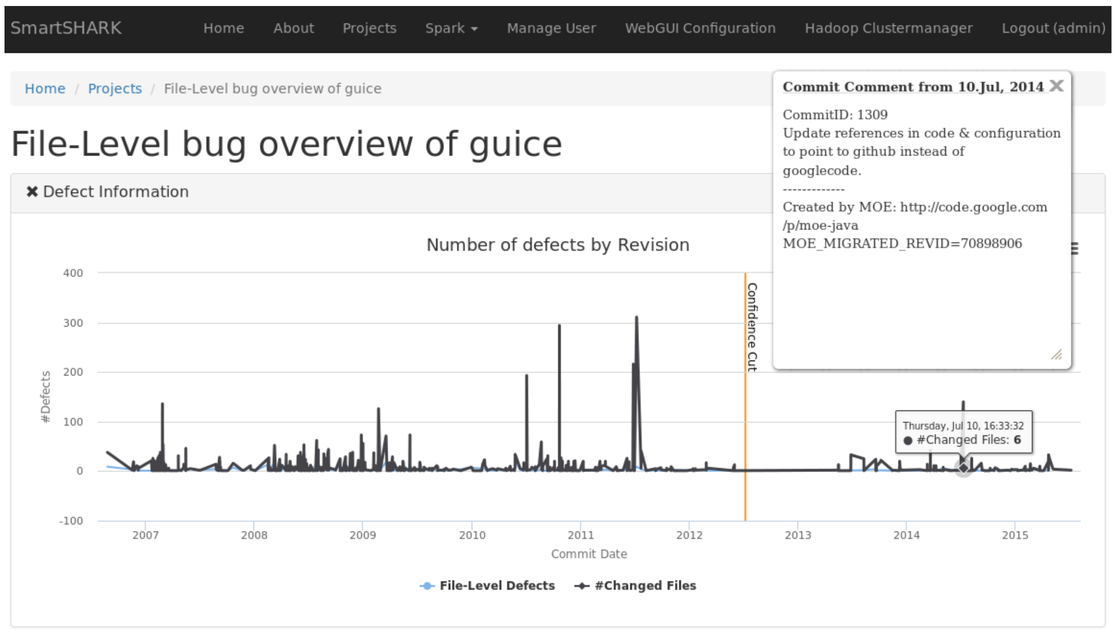


Figure 5.4.: Evaluation of the performance of the analysis process using Apache Spark for the chosen projects. The figure depicts the seconds till the classifier were build and evaluated by Apache Spark. This time is put into relation to the projects number of files.

The collections, which were newly created due to our Spark programs, can be read out by the web interface. What classifier are available for a project is illustrated in the project grid in the column "prediction collections" (see Section 4.2.1). To show how our platform can be used for reporting tasks, we implemented several functions:

- Zoomable graphs that depict the regions of the project used for training and testing. Furthermore, these graph show how many software artifacts are actually defect-prone according to the data, and how many are predicted as defect-prone, as well as the differences between the prediction and the actual values.
- We visualize a confidence cut, i.e., the line after which we are not sure if the bug labels in the data are correct, due to a small number of commits afterwards. Tan et al. [83] suggest, that this cut is chosen in a way, that the time from the cut till the last commit is the average bug fixing time. As this data is currently not available, we have chosen a time span of three years based on [58]. Additionally, the platform displays what data was used for training the classifier and from which commit the classification starts.



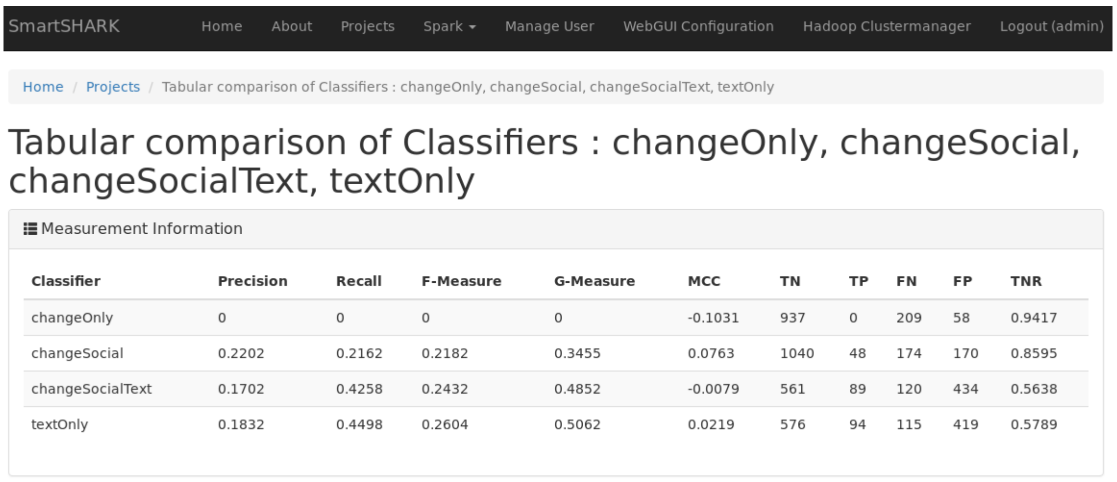
Commit Comment from 10.Jul, 2014

CommitID: 1309
Update references in code & configuration to point to github instead of googlecode.

Created by MOE: http://code.google.com/p/moe-java
MOE_MIGRATED_REVID=70898906

Figure 5.5.: *Overview of file-Level bugs for the project guice. The x-axis shows the commit date, whereas the y-axis illustrates the number of defects. There are two graphs in the figure: The black graph shows the number of changed files, whereas the blue graph illustrates the number of defects per revision. Furthermore, the commit message of the commit number 1309 is shown, because the user clicked on the corresponding point in the graph.*

- Classifier can be compared with each other. Either, if more than two classifier are chosen in tabular form (see Figure 5.6), or if two classifier are chosen in graphical form (see Figure 5.7). The table displays different important metrics, which are often used to evaluate defect prediction classifier (e.g. precision, recall). The graph comparison shows the defects per revision, together with number of defects predicted by the classifier.
- An overview of the changed files per revision and the number of bug-prone files is visualized by the platform. The number of bug-prone files is calculated by summing up the files per revision, where the bug label assigned by DECENT (see Section 2.1) is true. Figure 5.5 depicts the graph resulting from this calculation for the project "guice".


SmartSHARK
Home
About
Projects
Spark
Manage User
WebGUI Configuration
Hadoop Clustermanager
Logout (admin)
Home / Projects / Tabular comparison of Classifiers : changeOnly, changeSocial, changeSocialText, textOnly

Tabular comparison of Classifiers : changeOnly, changeSocial, changeSocialText, textOnly

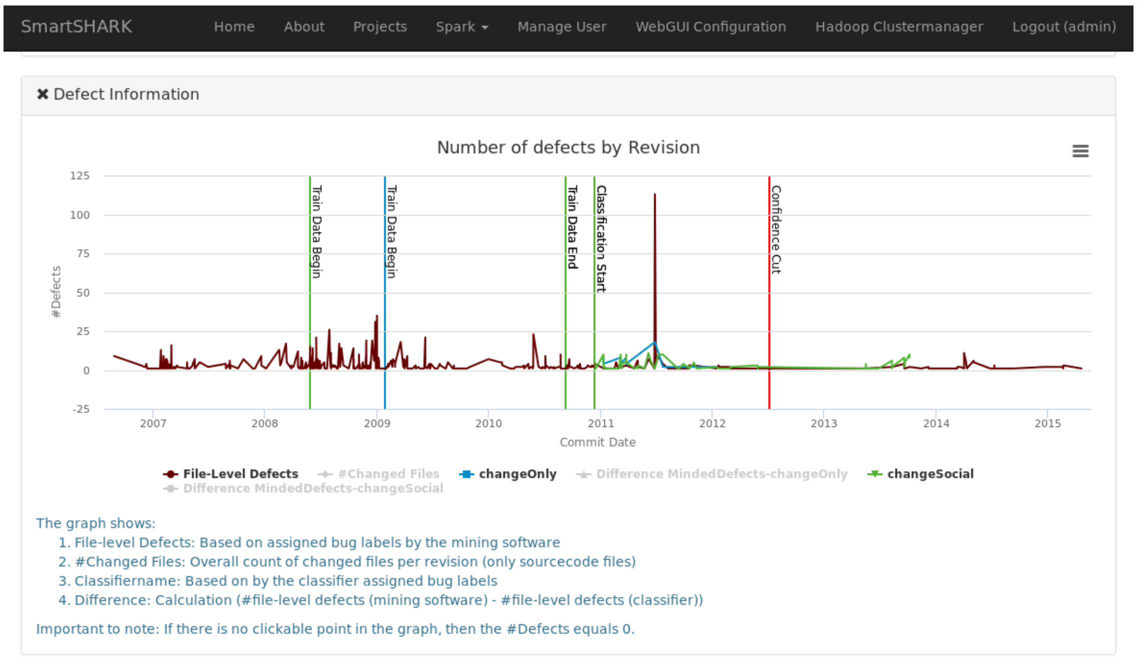
Measurement Information

Classifier	Precision	Recall	F-Measure	G-Measure	MCC	TN	TP	FN	FP	TNR
changeOnly	0	0	0	0	-0.1031	937	0	209	58	0.9417
changeSocial	0.2202	0.2162	0.2182	0.3455	0.0763	1040	48	174	170	0.8595
changeSocialText	0.1702	0.4258	0.2432	0.4852	-0.0079	561	89	120	434	0.5638
textOnly	0.1832	0.4498	0.2604	0.5062	0.0219	576	94	115	419	0.5789

Figure 5.6.: Tabular comparison of different classifier. This Table illustrates the different used classifier for the project guice and their evaluation metrics.

The results of our created classifier are illustrated in figures 5.8 - 5.12. As evaluation metrics, our platform calculate, amongst others, precision (see Figure 5.8), recall (see Figure 5.9), F-Measure (see Figure 5.10), G-Measure (see Figure 5.11), and MCC [14] (see Figure 5.12). These figures show the projects on the x-axis and the corresponding evaluation measure on the y-axis. Furthermore, there are bars for each of the created classifier for each project.

The results can be improved, but as we have mentioned above, we only want to demonstrate the analytic capabilities of our platform with this example. Therefore, no effort was made to tune or improve our prediction results. We were able to successfully mine the chosen projects and train classifier on the mined data. Furthermore, our platform is able to display the different results. Hence, our platform can be successfully used to conduct defect prediction research. But our platform is not limited to defect prediction research, as the mined data can also be used for other research areas (e.g. software evolution).



The graph shows:

1. File-level Defects: Based on assigned bug labels by the mining software
2. #Changed Files: Overall count of changed files per revision (only sourcecode files)
3. Classifiername: Based on by the classifier assigned bug labels
4. Difference: Calculation (#file-level defects (mining software) - #file-level defects (classifier))

Important to note: If there is no clickable point in the graph, then the #Defects equals 0.

Figure 5.7.: *Graphical comparison of two different classifier. The x-axis shows the commit date, whereas the y-axis illustrates the number of defects. There are three graphs active in this figure: First the number of file-Level Defects in brown, second the number of defects predicted by CL1 in blue and the number of defects predicted by CL2 in green. Furthermore, the graph shows the different starts of the used training Data. The training data ends and the classification start is the same for both compared classifier.*

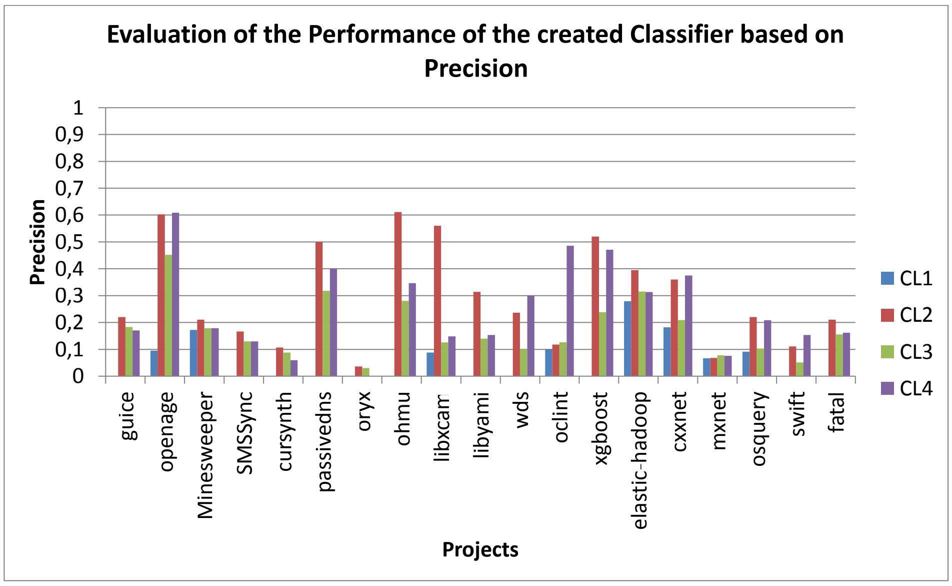


Figure 5.8.: *Evaluation of the performance of the created classifier based on precision. The figure depicts the different created classifier for each project and their precision.*

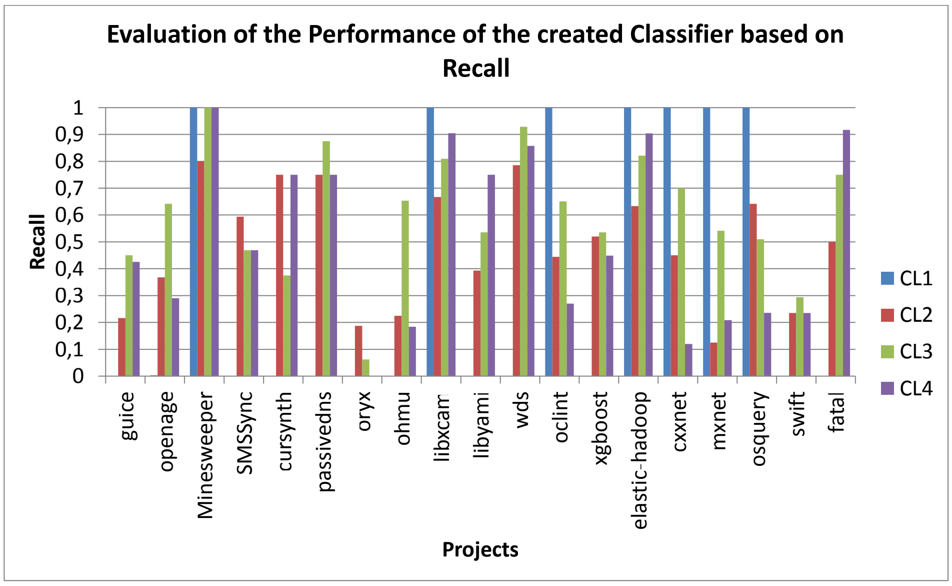


Figure 5.9.: *Evaluation of the performance of the created classifier based on recall. The figure depicts the different created classifier for each project and their recall.*

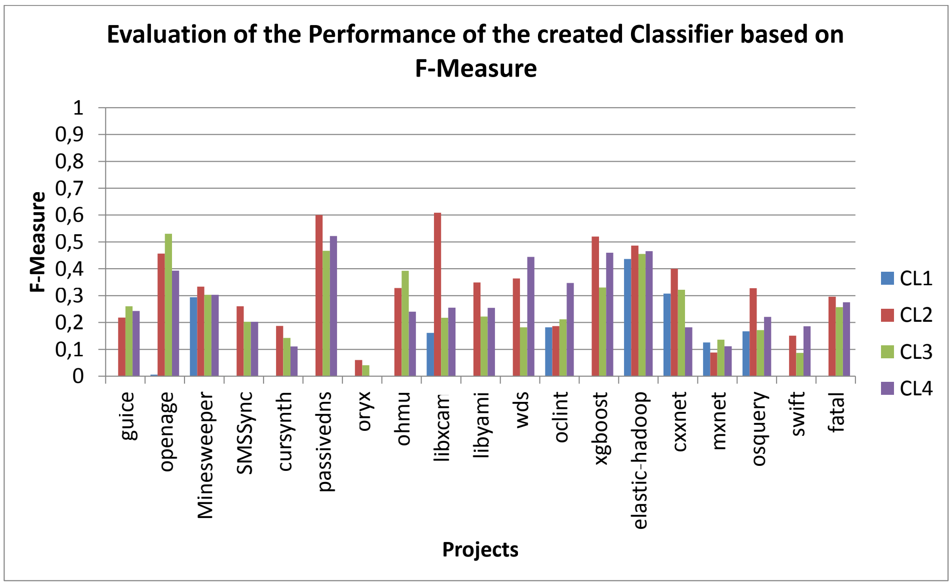


Figure 5.10.: Evaluation of the performance of the created classifier based on based on F-Measure. The figure depicts the different created classifier for each project and their F-Measure.

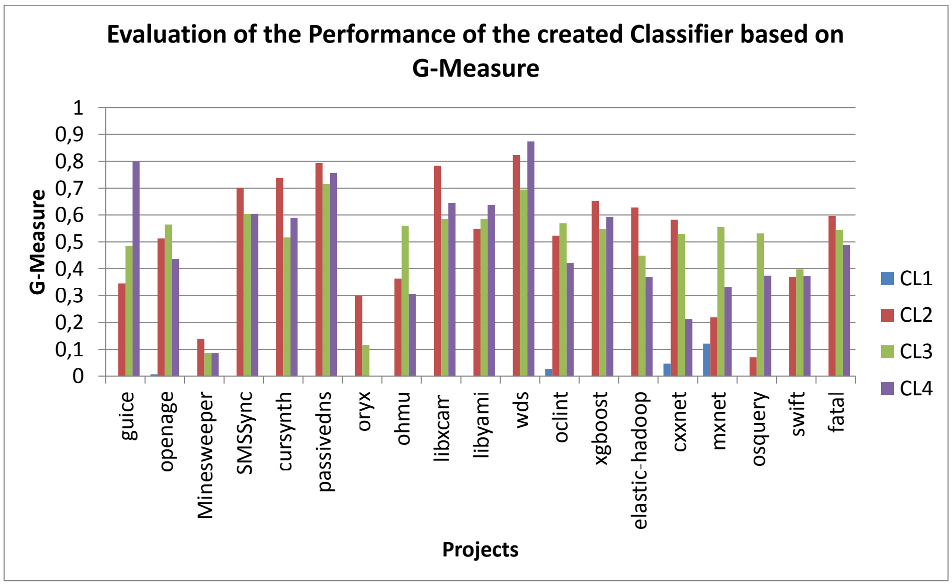


Figure 5.11.: Evaluation of the performance of the created classifier based on G-Measure. The figure depicts the different created classifier for each project and their G-Measure.

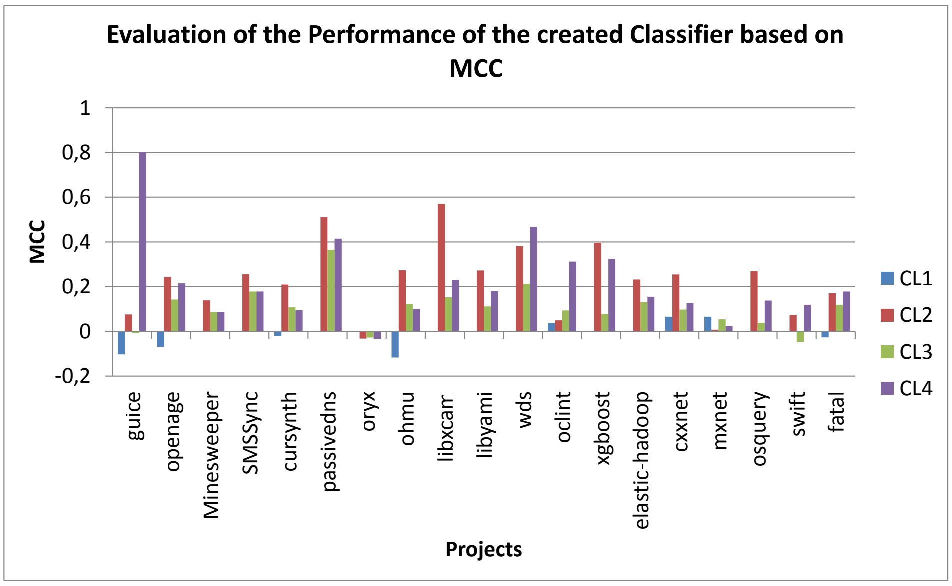


Figure 5.12.: Evaluation of the performance of the created classifier based on MCC. The figure depicts the different created classifier for each project and their MCC.

6. Results and Discussion

We have build a platform, where researchers are able to mine and analyse software projects. Furthermore, we use modern big data technologies like Apache Spark and Apache Hadoop for the analysis. Our platform provides a broad spectrum of functionalities:

- **Project management features**: Automatically mine a project, connect different data sources (e.g. metrics, duplication detection, see Section 4.3 and 4.2.1). Manage projects (delete, add).
- **Project analysis features**: Submit Spark jobs over the web GUI, manage Spark Jobs via the web GUI and retrieve the results. Usage of Apache Hadoop cluster for analysis in the background (see Section 4.4 and 4.2.2)
- **Additional features**: User management, backup management (see Section 4.2.4).
- **Cloud deployment**: The whole infrastructure can be set up and configured using our Vagrant and Ansible scripts (see Section 4.5), with minimal configuration effort.

Furthermore, we have given examples in our case studies on how to utilize our platform to facilitate defect prediction research (see Chapter 5). Additionally, we have described how the web interface of our platform can be used to directly access and compare the results of our defect prediction research. Different graphs and evaluation metrics are generated to give an overview of the performance of our classifiers (see Section 5.4). This shows, that our platform can be operationalized.

The goal was to create a platform with certain characteristics. An overview of these characteristics and how we implemented them can be found in Table 6.1. Table 6.1 and this thesis in general illustrate, that we succeeded in building a platform, which possess all the required characteristics. Important to note is the variety of data that the platform generates

Required Characteristic	Implementation
Easy to use	Web-based GUI based on Yii framework
Easy to deploy	Vagrant and Ansible scripts
Scalable analytics	Use of Apache Spark and Apache Hadoop, as well as MongoDB (Big Data technologies)
Flexible	Modular approach, based on EMF model instances (DECENT)
Extensible with new approaches	Web-based GUI based on Yii framework is easily extensible, adding of new meta models or metrics, which are then also combined in the DECENT model is possible
Usable for different research areas	Platform collects different kinds of data.

Table 6.1.: Platform requirements and their implementation in the proposed platform.

for further analysis. This data includes change, social and text metrics. This combination of different data makes the platform useful for different research directions.

Furthermore, we want to discuss the capabilities of our platform in this chapter. This includes design decision regarding the mining of the project data, why we have chosen Apache Spark as analytic backend and decisions regarding the cloud deployment.

Data Mining

The data mining is an important and powerful feature of our platform. Since it is responsible for the collection of the analysis data, it is a vital part of it. As we showed, we can collect the data of many projects without any problems and only experienced one failure of the mining (see Section 5.3). The model-based mining process based on DECENT provides a certain independence of the concrete tools used to collect the data. E.g., to substitute a tool, we only have to adapt the extraction of the DECENT facts from the raw assets created by the tool [66]. All other aspects of the mining remain untouched. Similarly, we can easily add new information sources to our platform. We just have to extend the DECENT meta-model with additional attributes that fit the structure of the new facts. E.g., if want to mine mailing lists, we first extract the relevant facts with a third party tool, e.g., MailingListStats [70]. Then, we map the facts to a corresponding meta-model and provide a model-to-model transformation, e.g., by means of ETL or EOL, which integrates the mined facts into the DECENT model and we are finished. The rest of the mining process remains untouched.

Software Analytics

As the analytics examples in Section 5.4 show, the analytics based on the mined data in the MongoDB with Apache Spark allows flexible and scalable analysis jobs. Java and Python are popular languages which should facilitate an easy adoption of Apache Spark and, by extension, of our platform. Moreover, Apache Spark provides good support for the connection with the MongoDB (e.g., through the deep framework, see Section 2.4). Moreover, there are already libraries for Apache Spark which are useful for the analysis of software projects, e.g., the *Mllib* for machine learning which we also use in our analytic examples, or *GraphX* for graph and graph-parallel computations. The choice of MongoDB as data back-end was made, as it is a state-of-the-art NoSQL database, often used in big data applications, scalable due to sharding (see Section 2.4) and provides a good tool support.

In our examples, we used only defect prediction and only data of one project. However, there is no reason why data from multiple projects cannot be used for the analytics, or why other interesting topics, like effort prediction or software evolution in general could not be studied.

Cloud Deployment

Our cloud deployment allows the scaling of the platform, which makes it suitable for analysing huge amounts of data, as well as executing highly resource-demanding analytics. Moreover, since the platform is readily available for researchers, they can concentrate on writing analytics programs and do not have to setup and maintain the analysis infrastructure. Due to our usage of automated deployment scripts, we facilitate that our platform can be easily adapted to both: new underlying software as well as a new cloud infrastructure. This also facilitates the recovery of our platform in case of fatal problems.

7. Limitations

At the actual state of development, the platform has its drawbacks, which we can classify into two different classes: Resource problems and program problems. Problems, which are contained in the first class, are problems that arise, because we only have limited resources in our lab. If there would be more resources available, these problems would not longer exist. The second class describe problems, which occur, because of the programs we use. Table 7.1 gives an overview of the different problems and their classes, as well as a possible solution to it.

The unreliability of InFamix (1) can only be solved, if another tool than InFamix is used, as it is not open source. Unfortunately, to our knowledge, there is no ready-to-use tool, which provides this set of metrics for different languages, like InFamix does. Nevertheless, there are tools in development which might be interesting (e.g. LISA [5]).

The problem, that InFamix is slow (2) can be solved by massive parallelization. DECENT provides an "FX" version of InFamix, where it is possible to distribute the different revisions, from which the metrics are calculated, between several nodes.

Since we currently only execute InFamix once, this also means that we currently can also only calculate software metrics for one programming language within our platform. Hence, projects with multiple programming languages are not yet supported (3). The solution for this is to extend the configuration possibilities of our platform and allow the selection of multiple programming languages. We can then execute InFamix once for each programming language. However, InFamix only supports Java, C, and C++ and we, therefore, might need to consider other tools for different languages.

Moreover, we currently cannot update the mined data of projects, because of limitations in the model-based mining framework (4). A workaround is to delete a project and re-

Problem Description	**Class**	**Possible Solution**
1) InFamix unreliable (sometimes wrong results or crashes)	Program	Substitute InFamix
2) InFamix slow	Program / Resource	Massive parallelization
3) Multiple programming languages within a project not yet supported	Program / Resource	Substitute InFamix or execute InFamix multiple times
4) No incremental update of projects	Program	Adapt mining process
5) Whole EMF model must fit into RAM during mining	Resource	Provide more resources or implement EMF-Fragments (see [80])
6) Bug labels are assigned heuristically	Program	Integrate ITS facts
7) Apache Spark does not provide the same amount of functionalities as Weka[50]	Program	Development of a framework based on Apache Spark
8) Web GUI slow in displaying large amount of data	Resource	Provide more resources
9) Due to high CPU and RAM demand, only one project can be mined at a time	Resource	Provide more resources
10) DECENTMongoDBConverter slow due to double pass to set cross references	Program	Refactor the DECENT-MongoDBConverter to use Apache Spark

Table 7.1.: Platform problems and possible solutions.

collect the data at a later point. However, this does not scale well, especially with larger projects. Instead, the mining process should be adapted to allow incremental mining.

The inherent limitation of EMF models, that the whole EMF model must fit into the RAM during the mining process (5), can be circumvented by either providing more resources or implementing the approach proposed by Markus Scheidgen (see [80]). Scheidgen developed an approach, which fragments EMF models. The models can be accessed like before, independent of the location of the fragment, which can be, e.g., saved in a MongoDB.

The assignment of bug labels (6) is another problem, because DECENT bases the decision of if an artifact is bug-prone or not only on the *BugFixMessage* extension of CVSAnaly and therefore the assignment is heuristically. CVSAnaly marks revisions as bugfix revisions by looking in the commit message and searching for keywords like *bug* or *fix*. In the processing of the project, an origin analysis is performed to find the revisions of the artifacts, which are responsible for this bugfix. Nevertheless, this information is sometimes error-prone or missing (e.g., a developer fixed a bug but does not mention it in the commit message). A partial solution for this problem would be the implementation of the BZExtractor, proposed by Makedonski et al. [66], to improve the bug label assignment.

Furthermore, Apache Spark does not provide the same functionalities as, e.g., Weka[50] (7). This can be solved by the development of a framework for Apache Spark, which has its focus on software project analysis.

The last two resource problems are, that only one project can be mined at a time (9) and that the web interface is slow at displaying a large amount of data (8). These problems can be circumvented by providing more resources in terms of CPUs and RAM.

The DECENTMongoDBConverter issue (10) is another program class problem. The converter is slow in the current implementation, as it needs to double pass the data to set cross references. Only a refactoring of the Converter and/or the use of Apache Spark might solve this problem by making the whole process faster

We are aware of the fact, that our actual infrastructure is not optimal. Figure 7.1 illustrates, how an optimized infrastructure can look like. To solve the problems with the webserver, we introduce a webserver cluster with one or more load balancer. Furthermore, instead of performing the mining on the webserver itself, we deploy a mining cluster. This mining cluster has a master node, which distributes the different mining tasks to the mining nodes. Additionally, this mining cluster communicates with the Apache Hadoop cluster via Apache Spark (e.g., for executing the DECENTMongoDBConverter). The Apache Hadoop cluster consists of the NameNode and the secondary NameNode, as well as the ResourceManager and a specified amount of Slaves. The webserver can communicate via Apache Spark with the Apache Hadoop cluster. Furthermore, instead of running the MongoDB on the webserver, we introduce the MongoDB cluster. The MongoDB cluster uses sharding [72] to make read and write accesses more efficient. Therefore, we deploy several shards (store the data), router (processes target operations / queries to shards) and three config servers (store the clusters metadata). The MongoDB documentation highlights, that exactly three config server are needed [72]. Figure 7.1 does not show, that all these clusters are deployed in a cloud environment, but this is strongly recommended as the addition of new resources to the machines is easier this way. Furthermore, the communication flow in the Apache Hadoop cluster is not depicted. This infrastructure could be easily set up via Vagrant and Ansible by adapting our scripts. The reason, why we do not have deployed our platform this way is our limitation on resources.

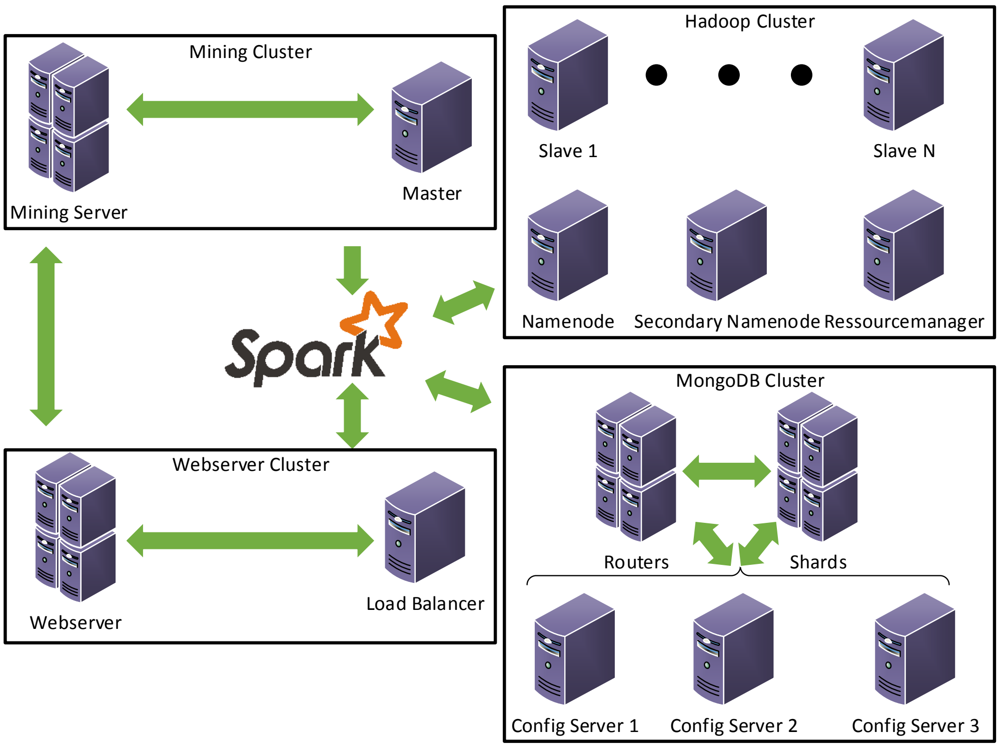


Figure 7.1.: Desired deployment of our platform. It is important to note, that the figure does not show, that every VM is deployed in a cloud environment. But this is strongly recommended. Furthermore, the communication flow in the Apache Hadoop cluster is not shown.

8. Related Work

The research on software mining and software analytics covers many different directions and aspects. However, most related work only considers software mining (e.g., [16, 23, 42, 57]) or analytic problems (e.g., defect prediction [19] or effort prediction [59]). Only very few of them consider not only mining or analytics, but actually try to combine these two aspects. Within this section, we focus only on platforms that combine the data mining with at least basic functions for analytics or already perform analytics.

Dyer et al. [32, 33] developed *Boa*, a domain-specific language and infrastructure for analyzing ultra-large-scale software repositories. It is a query system, where complicated queries can be executed to extract information from previously cached repositories using a distributed MapReduce query language. Boa programs compile down to different MapReduce jobs, which are then executed on an Apache Hadoop cluster. The key difference between Boa and our platform is the type of analysis that is supported. While Boa provides Abstract Syntax Trees (ASTs) of the projects, it does not directly enable deep analytics, e.g., based on software metrics, social aspects, or similar, which are possible with our platform. Such data would have to be calculated manually for each project by the researchers, which is a great inhibition when it comes to applications such as defect prediction or effort prediction. Moreover, this would probably lead to performance problems of the analytic approaches. Furthermore, Boa heavily uses MapReduce for its queries, whereas our platform uses Apache Spark. It is reported that MapReduce is inefficient for interactive analytics [92] as they are intended to be performed with Boa. This is the main problem that should be resolved by Apache Spark.

Gousios and Spinellis [49] developed the *Alitheia Core* platform to perform "large-scale software quality evaluation studies" [49]. The architecture of Alitheia Core is divided into three different layers: (1) Result presentation, (2) system core, and (3) data mirroring, storage, and retrieval. The first layer is implemented via a web front-end. The second layer includes a job scheduler and cluster service as well as other services, which are connected via an Open Services Gateway Initiative (OSGi) interface. The third layer is responsible for the storage and retrieval of the mined data. This platform provides a metrics plug-in system, which enables researchers to implement their own plug-ins to calculate metrics out of the mined data. From a structural perspective and general idea, Alitheia Core is very similar to our approach: we also have a mining part, an analytic core, and a web front-end. However, our platform is designed around big data technologies to allow scalable analytics. Moreover, our platform is deployed in the cloud in order to allow elastic scaling of resources. Finally, our analytic core, using Apache Spark, is more powerful in terms of computational capabilities due to the usage of an Apache Hadoop cluster for job execution and provides powerful algorithms for the data analysis through Apache Spark's Mllib and GraphX libraries.

A proprietary and not publicly available approach for building a software analytics platform with integrated mining is the Microsoft internal CODEMINE [24]. CODEMINE is quite similar to our platform in its general structure: the data is collected from repositories, stored in a flexible database and exposed via a single API on which analysis and tools can be defined. However, no details on the implementation of the platform are available and we assume that it is tailored to the Microsoft internal tooling environment. In addition to a high-level description of CODEMINE, the authors also provide lessons learned for building tools similar to CODEMINE, e.g., flexibility in the analysis and the data storage, separating the storage for each project, and hosting the platform in a cloud environment. We used these lessons learned for the design of our platform.

Perl et al. [75] developed VCCFinder, a tool which is used to find potential vulnerabilities in OSS projects. They extracted commits from different GitHub project repositories and created a dataset, which maps Common Vulnerabilities and Exposures (CVE) to Vulnerability-Contributing Commits (VCCs). Out of these commits, different features are extracted to train a classifier, which is used to predict, if a new commit introduces a vulnerability. Therefore, their approach mines data from GitHub and use this data later on for an analysis. There are several differences to our approach. First, our platform is designed as a general-purpose platform. Therefore, we do not fix ourselves to certain features, but try to mine as much data as possible. Hence, there are more analytic possibilities. Furthermore, as our platform calculates more complex metrics, we need more computational resources. We solved this by deploying our platform in a cloud environment in order to allow an elastic scaling of resources. Additionally, we use an Apache Hadoop cluster and other big data technologies for a fast analysis process, as some analysis (e.g. Cross-Project Defect Prediction (CPDP)) can be very time-consuming.

There are also commercial approaches that try to combine software mining with software analytics. Bitergia [17] offers several packages, which differ in the level of analysis capabilities. However, the analyses provided by Bitergia are on the level of BI, i.e., reporting of what happened in the repositories. E.g., they provide dashboards that display the number of performed commits or how many authors are active in the analysed project. Similar to Bitergia is the OpenHub project [18]. OpenHub is an open platform, where every user can add or edit projects. The platform calculates different statistics for added projects, which are similar to the statistics calculated by Bitergia and also on the BI level. In comparison to our platform, Bitergia and OpenHub do not support deep analytics or predictions for the future of projects. Additionally, users of Bitergia and OpenHub are not allowed to create their own analytics.

9. Conclusion and Outlook

We have developed a platform, which incorporates the mining and analysis of software projects. This platform makes use of modern big data technologies like Apache Spark, Apache Hadoop and MongoDB. Furthermore, this platform is easy to use and to deploy, because of the provided web-based interface and the deployment scripts based on Vagrant and Ansible. Due to the use of Apache Spark and MongoDB, as well as an Apache Hadoop cluster for the analysis of software projects, our platform is fast and scalable. Additionally, our platform is flexible and extensible, because of the use of DECENT for the project mining as well as the use of the Yii framework for the development of the web interface. Furthermore, we collect a variety of data by using DECENT. Hence, our platform is useful in conducting research in a broad range of research areas. Additionally, due to the extensibility of our platform, more data sources can be added easily.

In Chapter 2 we describe the foundations of our work, which includes the DECENT model, Apache Hadoop and Apache Spark, MongoDB, the Yii framework, Vagrand and Ansible. We design our approach in Chapter 3. This includes the analysis of the current situation (how researchers analyse software projects at the moment), an overview of our platform, the mining and analysis process and a description of the design of our cloud deployment. Furthermore, we illustrate our implementation in Chapter 4. We describe our platform in detail, as well as our web-based GUI and how the projects are mined and analysed. Furthermore, we describe our cloud deployment approach in detail. To evaluate our platform, we performed case studies, which are described in Chapter 5. These case studies show, that our platform successfully mine the projects. Furthermore, we give an application example by conducting defect prediction research on our test projects. The results illustrate, that our platform is valuable for conducting defect prediction research. Nevertheless, our classifier results can be improved, but this is not part of this master thesis. Our overall results are discussed in Chapter 6. Several problems, drawbacks and limitations of our platform are explained in Chapter 7. In the last chapter (see Chapter 8), we have a look at the related work and explain, why the existing work is not sufficient for the analysis of software projects.

As there are currently some problems, there is room for improving the platform. We have given some hints for improvements in Chapter 7. First, the platform should support incremental processing, as the current workaround (deleting and mining the project again) is time-intensive. Furthermore, as Java and Python are not the only languages that are used in software project analysis, another goal is to support more languages than just these two (e.g. R). What is also missing at the moment is a case study with a large project (e.g. Firefox) to evaluate our platform. But this is not possible with the current deployment of our platform. Furthermore, the performance of the mining should be improved. This includes the refactoring of the DECENTMongoDBConverter and changing the deployment of our platform, like it is explained in Chapter 7. Additionally, we plan to enhance the mining process by incorporating more tools, e.g., BZExtractor [66] to include data from Bugzilla or MailingListStats[1] to incorporate mailing lists. Furthermore, we could improve the performance of the mining as well as the analysis process, by using the sharding functionality of MongoDB. Hence, the queries can be processed in parallel and therefore the performance raises. Another problem at the moment is the performance of the created defect prediction models (see Section 5.4.4). Hence, we can test new attributes and other combinations of calculated metrics to get better evaluation results in terms of, e.g., precision and recall.

[1]See: `https://github.com/MetricsGrimoire/MailingListStats`.

Bibliography

[1] 01org, *Libxcam GitHub*, `https://github.com/01org/libxcam`, [accessed 28-August-2015].

[2] ——, *Libyami GitHub*, `https://github.com/01org/libyami`, [accessed 28-August-2015].

[3] ——, *Wds GitHub*, `https://github.com/01org/wds`, [accessed 28-August-2015].

[4] V. Abramova and J. Bernardino, "Nosql databases: mongodb vs cassandra", in *Proceedings of the International C* Conference on Computer Science and Software Engineering*, ACM, 2013, pp. 14–22.

[5] C. V. Alexandru and H. C. Gall, "Rapid multi-purpose, multi-commit code analysis", pp. 635–638, 2015.

[6] Ansible Inc., *Ansible Documentation*, `http://www.ansible.com/get-started`, [Online; accessed 26-July-2015], 2015.

[7] ——, *Intro to Playbooks*, `https://docs.ansible.com/ansible/playbooks_intro.html`, [Online; accessed 26-July-2015], 2015.

[8] Apache Software Foundation, *Apache Hadoop*, `https://hadoop.apache.org/`, [Online; accessed 24-July-2015], 2014.

[9] ——, *HDFS Architecture*, `https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`, [Online; accessed 24-July-2015], 2014.

[10] ——, *Spark 1.4 Documentation*, `https://spark.apache.org/docs/latest/programming-guide.html`, [Online; accessed 25-July-2015], 2015.

[11] ——, *Spark Cluster Mode Overview*, `https://spark.apache.org/docs/latest/cluster-overview.html`, [Online; accessed 25-July-2015], 2015.

[12] ——, *Spark Research*, `https://spark.apache.org/research.html`, [Online; accessed 25-July-2015], 2015.

[13] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits", in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, ACM, 2010, pp. 97–106.

[14] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview", *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.

[15] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Towards improving statistical modeling of software engineering data: think locally, act globally!", *Empirical Software Engineering*, vol. 20, no. 2, pp. 294–335, 2015.

[16] J. Bevan, E. J. Whitehead Jr, S. Kim, and M. Godfrey, "Facilitating software evolution research with kenyon", in *ACM SIGSOFT Software Engineering Notes*, ACM, vol. 30, 2005, pp. 177–186.

[17] Bitergia, *Bitergia*, `http://bitergia.com/`, [accessed 28-August-2015].

[18] Black Duck Software, Inc., *Open HUB*, `https://www.openhub.net/`, [accessed 28-August-2015].

[19] C. Catal and B. Diri, "A systematic review of software fault prediction studies", *Expert systems with applications*, vol. 36, no. 4, pp. 7346–7354, 2009.

[20] P. P.-S. Chen, "The entity-relationship model-toward a unified view of data", *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.

[21] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design", *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, 1994.

[22] Cloudera, *Oryx GitHub*, `https://github.com/cloudera/oryx`, [accessed 28-August-2015].

[23] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: a project memory for software development", *Software Engineering, IEEE Transactions on*, vol. 31, no. 6, pp. 446–465, 2005.

[24] J. Czerwonka, N. Nagappan, and W. Schulte, "CODEMINE: Building a Software Development Data Analytics Platform at Microsoft", *IEEE Softw.*, vol. 30, no. 4, pp. 64–71, 2013.

[25] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, "Analysing software repositories to understand software evolution", in *Software Evolution*, Springer, 2008, pp. 37–67.

[26] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[27] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis", in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, ACM, 2013, pp. 13–20.

[28] Distributed Machine Learning Common, *Cxxnet GitHub*, `https://github.com/dmlc/cxxnet`, [accessed 28-August-2015].

[29] ——, *Mxnet GitHub*, `https://github.com/dmlc/mxnet`, [accessed 28-August-2015].

[30] ——, *Xgboost GitHub*, `https://github.com/dmlc/xgboost`, [accessed 28-August-2015].

[31] U. Draisbach and F. Naumann, "Dude: the duplicate detection toolkit", in *Proceedings of the International Workshop on Quality in Databases (QDB)*, vol. 100000, 2010, p. 10 000 000.

[32] R. Dyer, H. A. Nguyen, H. Rajan, and T. Nguyen, "Boa: Ultra-Large-Scale Software Repository and Source Code Mining", *ACM Trans. Softw. Eng. and Methodology*, vol. forthcoming, 2015.

[33] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: a language and infrastructure for analyzing ultra-large-scale software repositories", in *Proc. IEEE/ACM 35th Int. Conf. on Softw. Eng.*, 2013.

[34] Eclipse Foundation, *Eclipse Modeling Framework (EMF)*, `https://eclipse.org/modeling/emf/`, [Online; accessed 24-July-2015], 2015.

[35] Elasticsearch BV, *Elasticsearch-hadoop GitHub*, `https://github.com/elastic/elasticsearch-hadoop`, [accessed 28-August-2015].

[36] Facebook Inc., *Fatal GitHub*, `https://github.com/facebook/fatal`, [accessed 28-August-2015].

[37] ——, *Osquery GitHub*, `https://github.com/facebook/osquery`, [accessed 28-August-2015].

[38] ——, *Swift GitHub*, `https://github.com/facebook/swift`, [accessed 28-August-2015].

[39] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens, "What does it take to develop a million lines of open source code?", in *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009, pp. 170–184.

[40] E. Fjellskål, *Passivedbs GitHub*, `https://github.com/gamelinux/passivedns`, [accessed 28-August-2015].

[41] M. Fotache, D. Cogean, *et al.*, "Nosql and sql databases for mobile applications. case study: mongodb versus postgresql", *Informatica Economica*, vol. 17, no. 2, pp. 41–58, 2013.

[42] D. M. German, "Mining cvs repositories, the softchange experience", *Evolution*, vol. 245, no. 5,402, pp. 92–688, 2004.

[43] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models", in *Proc. of the 37th Int'l Conf. on Software Engineering (ICSE)*, 2015.

[44] M. Godfrey and Q. Tu, "Tracking structural evolution using origin analysis", in *Proceedings of the international workshop on Principles of software evolution*, ACM, 2002, pp. 117–119.

[45] M. W. Godfrey and Q. Tu, "Evolution in open source software: a case study", in *Software Maintenance, 2000. Proceedings. International Conference on*, IEEE, 2000, pp. 131–142.

[46] Google, *Guice GitHub*, `https://github.com/google/guice`, [accessed 28-August-2015].

[47] ——, *Ohmu GitHub*, `https://github.com/google/ohmu`, [accessed 28-August-2015].

[48] M. Göttsche, “Automated Deployment and Distributed Execution of Scientific Software in the Cloud using DevOps and Hadoop MapReduce”, Master’s thesis, Apr. 2015.

[49] G. Gousios and D. Spinellis, “Alitheia core: an extensible software quality monitoring platform”, in *Proceedings of the 31st International Conference on Software Engineering*, IEEE Computer Society, 2009, pp. 579–582.

[50] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update”, *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009, ISSN: 1931-0145. DOI: `10.1145/1656274.1656278`.

[51] HashiCorp, *Vagrant Documentation*, `https://docs.vagrantup.com/v2/`, [Online; accessed 26-July-2015], 2015.

[52] A. E. Hassan and R. C. Holt, “The top ten list: dynamic fault prediction”, in *Software Maintenance, 2005. ICSM’05. Proceedings of the 21st IEEE International Conference on*, IEEE, 2005, pp. 263–272.

[53] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set”, *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[54] S. Herbold, “Crosspare: a tool for benchmarking cross-project defect predictions”, in *Proc. 4th Int. Workshop on Software Mining (SoftMine)*, 2015.

[55] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, “Forecasting the number of changes in eclipse using time series analysis”, in *Mining Software Repositories, 2007. ICSE Workshops MSR’07. Fourth International Workshop on*, IEEE, 2007, pp. 32–32.

[56] V. Honsel, D. Honsel, and J. Grabowski, “Software process simulation based on mining software repositories”, in *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, IEEE, 2014, pp. 828–831.

[57] J. Howison, M. S. Conklin, and K. Crowston, “Ossmole: a collaborative repository for floss research data and analyses”, in *1st International Conference on Open Source Software, Genova, Italy*, 2005, pp. 11–14.

[58] T. Jiang, L. Tan, and S. Kim, “Personalized defect prediction”, in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, IEEE, 2013, pp. 279–289.

[59] M. Jorgensen and M. Shepperd, “A systematic review of software development cost estimation studies”, *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2007, ISSN: 0098-5589. DOI: `10.1109/TSE.2007.256943`.

[60] H. Kagdi, M. L. Collard, and J. I. Maletic, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution”, *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, 2007.

[61] S. Kim, E. J. Whitehead Jr, and Y. Zhang, “Classifying software changes: clean or buggy?”, *Software Engineering, IEEE Transactions on*, vol. 34, no. 2, pp. 181–196, 2008.

[62] S. Kim, H. Zhang, R. Wu, and L. Gong, “Dealing with noise in defect prediction”, in *Software Engineering (ICSE), 2011 33rd International Conference on*, IEEE, 2011, pp. 481–490.

[63] S. Kumaresh and R. Baskaran, “Mining software repositories for defect categorization.”, *Journal of Communications Software & Systems*, vol. 11, no. 1, 2015.

[64] E. Lawlor, *HackerNews GitHub*, `https://github.com/lawloretienne/HackerNews`, [accessed 28-August-2015].

[65] ——, *Minesweeper GitHub*, `https://github.com/lawloretienne/Minesweeper`, [accessed 28-August-2015].

[66] P. Makedonski, F. Sudau, and J. Grabowski, “Towards a model-based software mining infrastructure”, *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1–8, 2015.

[67] T. Mende, “Replication of defect prediction studies: problems, pitfalls and recommendations”, in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ACM, 2010, p. 5.

[68] T. Mens, J. Fernández-Ramil, and S. Degrandsart, “The evolution of eclipse”, in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, IEEE, 2008, pp. 386 –395.

[69] T. Menzies, M. Rees-Jones, R. Krishna, and C. Pape, *The promise repository of empirical software engineering data*, `http://openscience.us/repo`. North Carolina State University, Department of Computer Science, 2015.

[70] Metrics Grimoire, *MalingListStats GitHub*, `https://github.com/MetricsGrimoire/MailingListStats`, [accessed 28-August-2015].

[71] MongoDB Inc., *MongoDB Drivers*, `http://docs.mongodb.org/ecosystem/drivers/`, [Online; accessed 25-July-2015], 2015.

[72] ——, *MongoDB Sharding Introduction*, `http://docs.mongodb.org/manual/core/sharding-introduction/`, [Online; accessed 25-July-2015], 2015.

[73] ——, *Who's using MongoDB*, `https://www.mongodb.org/community/deployments`, [Online; accessed 25-July-2015], 2015.

[74] M. Peacock, *Creating Development Environments with Vagrant*. Packt Publishing Ltd, 2015.

[75] H. Perl, D. Arp, S. Dechand, S. Fahl, Y. Acar, F. Yamaguchi, K. Rieck, and M. Smith, "Vccfinder: finding potential vulnerabilities in open-source projects to assist code audits", in *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, ACM.

[76] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction", in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ACM, 2012, p. 61.

[77] J. Ramos, "Using tf-idf to determine word relevance in document queries", in *Proceedings of the first instructional conference on machine learning*, 2003.

[78] R. Saito, *Oclint GitHub*, `https://github.com/oclint/oclint`, [accessed 28-August-2015].

[79] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, "Class level fault prediction using software clustering", in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, IEEE, 2013, pp. 640–645.

[80] M. Scheidgen, "Reference representation techniques for large models", in *Proceedings of the Workshop on Scalability in Model Driven Engineering*, ACM, 2013, p. 5.

[81] SFTtech, *OpenAge GitHub*, `https://github.com/SFTtech/openage`, [accessed 28-August-2015].

[82] Y. Shin, A. Meneelay, L. William, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity as indicators of software vulnerabilities", *IEEE Trans. Softw. Eng.*, vol. 37, no. 6, pp. 772–787, 2011.

[83] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data", pp. 99–108, 2015.

[84] M. Tytel, *Cursynth GitHub*, `https://github.com/mtytel/cursynth`, [accessed 28-August-2015].

[85] Ushahidi, *SMSSync GitHub*, `https://github.com/ushahidi/SMSSync`, [accessed 28-August-2015].

[86] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, *et al.*, "Apache hadoop yarn: yet another resource negotiator", in *Proceedings of the 4th annual Symposium on Cloud Computing*, ACM, 2013, p. 5.

[87] Z. Wei-ping, L. Ming-Xin, and C. Huan, "Using mongodb to implement textbook management system instead of mysql", in *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, IEEE, 2011, pp. 303–305.

[88] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[89] Yii Software LLC., *About Yii*, `http://www.yiiframework.com/doc-2.0/guide-intro-yii.html`, [Online; accessed 26-July-2015], 2015.

[90] ——, *Application Structure Overview*, `http://www.yiiframework.com/doc-2.0/guide-structure-overview.html`, [Online; accessed 26-July-2015], 2015.

[91] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing", in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, 2012, pp. 2–2.

[92] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets", in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[93] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process", in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ACM, 2009, pp. 91–100.

[94] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse", in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, IEEE, 2007, pp. 9–9.

Appendices

A. Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"
SLAVES_COUNT = $count

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = 'dummy'
    config.ssh.username = 'root'
    config.ssh.private_key_path = $keypath)
    config.vm.synced_folder '.', '/vagrant'
    config.vm.boot_timeout = 600

    config.vm.provider :openstack do |os|
        os.openstack_auth_url = 'http://172.18.154.2:5000/v2.0/tokens'
        os.username           = $username
        os.password           = $password
        os.tenant_name        = $tenantname
        os.flavor             = $flavor
        os.image              = '6a95af53-c127-48fe-8b98-3f74562d1ad9'
        os.floating_ip_pool   = 'ext-net'
        os.keypair_name       = $keyname
    end

    config.vm.define "namenode" do |namenode|
        namenode.vm.hostname = "namenode"
    end

    SLAVES_COUNT.times do |i|
        config.vm.define "slave#{i+1}" do |slave|
            slave.vm.hostname = "slave#{i+1}"
        end
    end

    config.vm.define "webserver" do |webserver|
        webserver.vm.hostname = "webserver"
        webserver.vm.provider :openstack do |os|
          os.openstack_auth_url = 'http://172.18.154.2:5000/v2.0/tokens'
          os.username           = $username
          os.password           = $password
          os.tenant_name        = $tenantname
          os.flavor             = $flavor
          os.image              = '6a95af53-c127-48fe-8b98-3f74562d1ad9'
          os.floating_ip_pool   = 'ext-net'
          os.keypair_name       = $keyname
     end
    end

    config.vm.define "resourcemanager" do |resourcemanager|
        resourcemanager.vm.hostname = "resourcemanager"

        resourcemanager.vm.provision :ansible do |ansible|
            ansible.verbose = "v"
            ansible.sudo = true
            ansible.playbook = "deployment/site.yml"
            ansible.limit = "all"

            slaves = (1..SLAVES_COUNT).to_a.map {|id| "slave#{id}"}
            ansible.groups = {
                "NameNode" => ["namenode"],
                "ResourceManager" => ["resourcemanager"],
                "Slaves" => slaves,
                "Webserver" => ["webserver"]
            }
        end
    end

end
```

Listing A.1: Used Vagrantfile for the cloud deployment.

B. Apache Spark Analysis Program

```java
package de.smadap.casestudy;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.classification.LogisticRegressionModel;
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD;
import org.apache.spark.mllib.evaluation.MulticlassMetrics;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.bson.types.ObjectId;

import scala.Tuple2;

import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.MongoClient;
import com.mongodb.QueryBuilder;
import com.stratio.deep.commons.entity.Cell;
import com.stratio.deep.commons.entity.Cells;
import com.stratio.deep.core.context.DeepSparkContext;
import com.stratio.deep.mongodb.config.MongoConfigFactory;
import com.stratio.deep.mongodb.config.MongoDeepJobConfig;
import de.smadap.playbook.util.SparkUtils;
/**
 * Class, which reads out data from a mongodb and classifies it into bugprone
     and
 * non-bugprone using a change metric classifier
 *
 * Procedure:
 * 1) Create Spark Context with params like host, jobname, collection, etc.
 * 2) Create a MongoDeepConfig (query the correct fields)
 * 3) Create an RDD with the deep config to get the mongodata (query the
    mongodb)
 * 4) Get Sampled Training data (from trainingBegin to trainingEnd, which are
     both commitIDs)
 * 5) Preprocess the data: Create LabeledPoints (label, metrics vector)
 * 6) Create classifier
 * 8) Get Classification data (from ClassificationStart to classification end
    )
 * 9) Preprocess the data: Create a map with fileId:LabeledPoint, so that we
    can track the results to the correct file.
 * The labeledPoint is needed by the classifier and has a label and the
    metrics vector
 * 10) Create data for the prediction collection, which can be read out by
    the webserver
 * 11) Evaluate the classifier
 * 12) Add meta-data (like fp,fn,tp,tn) to the prediction collection
 * 13) Save the prediction collection
 *
 * Input:
 * <projectName> <trainingBeginCommitID> <trainingEndCommitID> <
    classificationStartCommitID> <classificationEndCommitID> <classifierName>
 *
 * It is important to note, that we only look at files, which have a label
    assigned to it. It can happen, that we have
 * files, which dont have a labled assigned. This is, because decent uses
    origin analysis for assigning bug labels. Therefore,
 * if the data is "too new" we dont have any clues if this file is bug prone
    or not.
 *
 * @author Fabian Trautsch
 *
 */

public class ChangeClassification {

    public static void main(String[] args) throws IOException {

      // Set up your jobname
      String job = "java:ChangeClassification";

      // Set you host, where mongodb is running
        String host = "172.18.135.221";
        int port = 27017;

        // Set the mongodb where you are connecting to
        String database = "smadap";
        String inputCollection = "file";
        String projectName = args[0];

        // Set the interval from which training should be done
      int trainingBegin = Integer.parseInt(args[1]);
      int trainingEnd = Integer.parseInt(args[2]);


      // Set commit id till when it should be classified
      int classificationStart = Integer.parseInt(args[3]);
      int classificationEnd = Integer.parseInt(args[4]);
      String classifierName = args[5];

      // Set name of the prediction collection
      String predictionCollection = "prediction_"+projectName+"_"+
    classifierName;

      // First delete collection
      MongoClient client = new MongoClient(host, port);
      client.getDB(database).getCollection(predictionCollection).drop();

      // Set predictionLAbels
      ArrayList<String> labels = new ArrayList<String>();
        labels.add("true");
        labels.add("false");
        // Create a spark config (if you test locally, set "-Dspark.master=
    local[2]" to your VM Arguments
        SparkConf conf = new SparkConf().setAppName(job);
      SparkContext sc = new SparkContext(conf);

        // Creating the Deep Context where argument is the spark context
        DeepSparkContext deepContext = new DeepSparkContext(sc);

        SparkUtils sparkUtils = new SparkUtils(deepContext);

        DBObject bsonFields = new BasicDBObject();
        bsonFields.put("metrics.HunkCount", 1);
        bsonFields.put("metrics.LOC", 1);
        bsonFields.put("metrics.FileSize", 1);
        bsonFields.put("metrics.LABELBugFixAverageWeight", 1);
        bsonFields.put("metrics.LinesRemoved", 1);
        bsonFields.put("metrics.LinesAdded", 1);
        bsonFields.put("_id", 1);

        JavaRDD<Cells> input = sparkUtils.doSampling(bsonFields, projectName,
     trainingBegin, trainingEnd,
            host, port, database, inputCollection);


        System.out.println(input.count());

        /*
         * Here we parse the data in a way, that we look at our metrics and
    create labeledpoint
         * instances out of it. For that, we need our label (0.0 or 1.0) and
    a vector (sparse or dense)
         *
         * This step is done in parallel!
         */
        JavaRDD<LabeledPoint> training = input
                .map(new Function<Cells, LabeledPoint>() {

                    @Override
                    public LabeledPoint call(Cells cells) throws Exception {
                        // Get metrics
                        Cells metrics = (Cells) cells.getCellByName("metrics").getCellValue();

                          // First: Get the label attribute
                          String label = metrics.getCellByName("LABELBugFixAverageWeight").getString();

                          // We need a double value for the label
                          double labelValue = getDoubleForLabel(label);


                          /*
                           * Now we need a double[] to create the labeledpoints. The
                           * fastest (but sometimes exhausting) way to achieve this is by
                           * adding the double values by hand.
                           *
                           * Another possibility is shown below
                           *
                           * Nevertheless, we need to convert this list back to primitive double array
                           * Additionally, we need to be careful with the sorting!
                           */

                          // We need to check if they exist. If not the value is 0.0!
                          double HunkCount = 0.0;
                          double LOC = 0.0;
                          double FileSize = 0.0;
                          double linesRemoved = 0.0;
                          double linesAdded = 0.0;

                          if(metrics.getCellByName("HunkCount") != null) {
                            HunkCount = Double.parseDouble(metrics.getCellByName("HunkCount").getString());
                          }
```

```java
                                        if(metrics.getCellByName("LOC") != null) {
                                          LOC = Double.parseDouble(metrics.getCellByName("LOC").getString());
                                        }

                                        if(metrics.getCellByName("FileSize") != null) {
                                          FileSize = Double.parseDouble(metrics.getCellByName("FileSize").getString());
                                        }

                                        if(metrics.getCellByName("LinesRemoved") != null) {
                                          linesRemoved = Double.parseDouble(metrics.getCellByName("LinesRemoved").getString());
                                        }

                                        if(metrics.getCellByName("LinesAdded") != null) {
                                          linesAdded = Double.parseDouble(metrics.getCellByName("LinesAdded").getString());
                                        }

                                        double[] values = {HunkCount, LOC, FileSize, linesAdded, linesRemoved};


                                        return new LabeledPoint(labelValue, Vectors.dense(values));
                                }
                        });

        training.cache();



        final LogisticRegressionModel model = new LogisticRegressionWithSGD().run(training.rdd());

        // Get classification data
        QueryBuilder classifyQuery = QueryBuilder.start();
        // We only want to have artifacts of this specific project with this confidence
        classifyQuery.put("projectName").is(projectName);
        classifyQuery.and("type").is("code");
        // We just want to have all files with commitID greater than 50
        classifyQuery.and("commitID").greaterThan(classificationStart).lessThan(classificationEnd);
        classifyQuery.and("metrics.LABELBugFixAverageWeight").in(labels);



        MongoDeepJobConfig<Cells> classifyEntitiy = MongoConfigFactory.createMongoDB().host(host).port(port).database(database)
                .collection(inputCollection)
                .createInputSplit(false)
                .filterQuery(classifyQuery).fields(bsonFields);





        // Create a javardd file out of the response for parallel processing
        JavaRDD<Cells> classify = deepContext.createJavaRDD(classifyEntitiy);


        /*
         * Here we parse the data in a way, that we look at our metrics and create labeledpoint
         * instances out of it. For that, we need our label (0.0 or 1.0) and a vector (sparse or dense)
         *
         * This step is done in parallel!
         */
        JavaRDD<Map<String, LabeledPoint>> classificationData = classify
                .map(new Function<Cells, Map<String, LabeledPoint>>() {

                    @Override
                    public Map<String, LabeledPoint> call(Cells cells) throws Exception {
                      // Get metrics
```

```
                        Cells metrics = (Cells) cells.getCellByName("metrics").
    getCellValue();

                        // First: Get the label attribute
                          String label = metrics.getCellByName("
    LABELBugFixAverageWeight").getString();

                          // We need a double value for the label
                          double labelValue = getDoubleForLabel(label);

                          double HunkCount = 0.0;
                          double LOC = 0.0;
                          double FileSize = 0.0;
                          double linesRemoved = 0.0;
                          double linesAdded = 0.0;

                          if(metrics.getCellByName("HunkCount") != null) {
                            HunkCount = Double.parseDouble(metrics.
    getCellByName("HunkCount").getString());
                          }

                          if(metrics.getCellByName("LOC") != null) {
                            LOC = Double.parseDouble(metrics.getCellByName("LOC
    ").getString());
                          }

                          if(metrics.getCellByName("FileSize") != null) {
                            FileSize = Double.parseDouble(metrics.getCellByName
    ("FileSize").getString());
                          }

                          if(metrics.getCellByName("LinesRemoved") != null) {
                            linesRemoved = Double.parseDouble(metrics.
    getCellByName("LinesRemoved").getString());
                          }

                          if(metrics.getCellByName("LinesAdded") != null) {
                            linesAdded = Double.parseDouble(metrics.
    getCellByName("LinesAdded").getString());
                          }
```

```
                    double[] values = {HunkCount, LOC, FileSize,
linesAdded, linesRemoved};

                    // Get _id
                    String id = ((ObjectId) cells.getCellByName("_id").
getCellValue()).toString();

                    Map<String, LabeledPoint> returnMap = new HashMap<
String, LabeledPoint>();
                    returnMap.put(id, new LabeledPoint(labelValue,
Vectors.dense(values)));

                    return returnMap;
                }
            });


    // Create prediction set for mongodb
    JavaRDD<Cells> idandlabel = classificationData.map(
      new Function<Map<String, LabeledPoint>, Cells>() {

        public Cells call(Map<String, LabeledPoint> map) {

          // Get labeledpoint
          String id = (String) map.keySet().toArray()[0];
          LabeledPoint point = map.get(id);
          Double prediction = model.predict(point.features());

          Cells result = new Cells();
          result.add(Cell.create("fileId", id));
          result.add(Cell.create("prediction",getLabelForDouble(
prediction)));

          return result;
        }
      }
    );
```

```
        // Compute raw scores on the test set.
        JavaRDD<Tuple2<Object, Object>> predictionAndLabels =
    classificationData.map(
          new Function<Map<String, LabeledPoint>, Tuple2<Object, Object>>() {
      public Tuple2<Object, Object> call(Map<String, LabeledPoint> map)
          throws Exception {
        // Get labeledpoint
              String id = (String) map.keySet().toArray()[0];
              LabeledPoint point = map.get(id);
              Double prediction = model.predict(point.features());
              return new Tuple2<Object, Object>(prediction, point.label());
      }
          }
        );

        // Get evaluation metrics.
        // As we see here, the ordering of the confusion matrix is:
        //  0 1
        //0
        //1
        MulticlassMetrics metrics = new MulticlassMetrics(predictionAndLabels
    .rdd());
        System.out.println("Labels  = "+metrics.labels()[0]);
        System.out.println("Labels  = "+metrics.labels()[1]);
        System.out.println("Confusion matrix = "+metrics.confusionMatrix());

        // Calculate tn, fn, fp, tp
        System.out.println("TN = "+metrics.confusionMatrix().toArray()[0]);
        System.out.println("FN = "+metrics.confusionMatrix().toArray()[1]);
        System.out.println("FP = "+metrics.confusionMatrix().toArray()[2]);
        System.out.println("TP = "+metrics.confusionMatrix().toArray()[3]);


        // Add needed metadata (needed by webserver)
        Cells cutOffData = new Cells();
        cutOffData.add(Cell.create("trainCommitIDStart", trainingBegin));
        cutOffData.add(Cell.create("trainCommitIDEnd", trainingEnd));
        cutOffData.add(Cell.create("classificationStart", classificationStart
  ));
        cutOffData.add(Cell.create("TN", metrics.confusionMatrix().toArray()
  [0]));
        cutOffData.add(Cell.create("TP", metrics.confusionMatrix().toArray()
  [3]));
        cutOffData.add(Cell.create("FN", metrics.confusionMatrix().toArray()
  [1]));
        cutOffData.add(Cell.create("FP", +metrics.confusionMatrix().toArray()
  [2]));

        List<Cells> output = idandlabel.collect();
        output.add(cutOffData);

        JavaRDD<Cells> toMongoDB = deepContext.parallelize(output);
        // Create outputjob config
        MongoDeepJobConfig<Cells> outputConfigEntity = MongoConfigFactory.
  createMongoDB().host(host).port(port).database(database)
                .collection(predictionCollection).initialize();

        // Save RDD
        DeepSparkContext.saveRDD(toMongoDB.rdd(), outputConfigEntity);

        // Stop the whole context
        deepContext.close();


    }

  protected static double getDoubleForLabel(String label) {
    if(label.equals("true")) {
      return 1.0;
    }
    return 0.0;
  }

  protected static String getLabelForDouble(Double label) {
      if(label.equals(1.0)) {
        return "true";
      }
```

```
        return "false";
    }


}
```

Listing B.1: Apache Spark analysis program example. Used in the Case Study for Defect Prediction.

C. Paper

SmartSHARK: Smart Software History Analysis and Retrieval of Knowledge

A smart data platform for software analytics

Fabian Trautsch, Steffen Herbold, Philip Makedonski, Jens Grabowski
Institute for Computer Science, Georg-August-Universität Göttingen, German
{trautsch,herbold,makedonski,grabowski}@cs.uni-goettingen.de

ABSTRACT

Nowadays, repository mining and software analytics together form a large research direction, with hundreds of publications on how data should be collected from repositories, with datasets that were donated by researchers, as well as many analyses performed on the data sets. However, for the most part the mining and analytic activities are independent of each other. With SmartSHARK, we want to introduce a solution that integrates the data collection through repository mining with a powerful analytics platform that enables deep analysis of software projects. SmartSHARK is developed as a cloud platform with a web front-end that provides researchers with an analytic sandbox. SmartSHARK provides different kinds of data, including source code metrics, social metrics as well as the textual differences between revisions on multiple levels of abstraction, e.g., the file level, the class level, and the method level. The mined data is stored in a performant NoSQL database and software analysis jobs can be defined with Apache Spark. We provide an example in which we show how SmartSHARK can be utilized to define powerful analytics that combine different aspects of the development of a software.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Experimentation, Algorithms

Keywords

software mining, software analytics, smart data

1. INTRODUCTION

During the last decade, repository mining and software analytics became a highly topical research topic that spawned hundreds of publications. The rise of this research area resulted in many tools for mining repositories, datasets mined by researchers, and analysis performed by researchers. However, for the most part independent of each other. There are multiple projects that provide integration points for the currently ongoing research. Data repositories like teraPROMISE [56] and GitHub archive [41] collect data and facilitate research on common data sets, but do not provide means for the data analysis. Projects like MetricsGrimoire try to gain control of the mining tools by giving them a common home. The commercial Bitergia [13] platform uses software from the MetricsGrimoire to provide Business Intelligence (BI) about projects. But Bitergia is a closed source solution and, moreover, does not support complex analysis, e.g., based on machine learning. Other platforms, such as CrossPare [45], focus only on analytics and completely omit the data mining.

From a data science perspective, all of the above approaches are lacking, due to the missing integration of data collection, storage, and analytics. Here, the ideal scenario is an analytic sandbox [27], which contains all the data, can be updated with new data, and allows complex analysis of the data. The creation of this infrastructure and the analysis of the mined data is difficult [12, 37, 56, 66]. In addition, creating an infrastructure that scales well is even harder to achieve [66]. Moreover, the data should be in control of the data scientist. If we map this to software analytics based on repository mining, it means we require a platform that integrates the mining of repositories with the means for software analytics.

With SmartSHARK (`http://smartshark.informatik.uni-goettingen.de`[1]), we want to provide an analytic sandbox for software analytics. SmartSHARK enables researchers to automatically mine data and provides them with powerful and scalable tooling for the analysis of the data. SmartSHARK mines the complete history of a project and collects static source code metrics, change metrics, social metrics, as well as the textual diffs [33] for each revision. Through the storage of the mined data in a shared database, SmartSHARK implicitly grows a repository with data about software projects that can be exploited by researchers. With MongoDB [59] as database back-end, we employ a state-of-the-art NoSQL database which can handle huge amounts of data without scalability problems. To facilitate analysis, SmartSHARK makes use of Apache Spark [8]. Through the definition of their own Spark jobs, researchers can perform different analyses on the projects. We show with a sample

[1]Credentials for reviewers: admin / 86t4$3RzP ; advanceduser / 6Xt4$15PQsT ; user / 5tx261Rz$

application, how SmartSHARK can be used to define analysis jobs, that even combine text analysis, and metric based software analytics.

Moreover, a platform that integrates mining and analysis is not only interesting for researchers, but also important to create actionable software analytics. Therefore, we modified the SmartSHARK front-end in a way that the results of our sample analysis are displayed directly together with the projects. Through this, reporting becomes part of the platform, which is the precursor for putting the software analytics into operation. Our approach is in this regard similar to Bitergia, but allows complex analytics instead of BI.

SmartSHARK is developed as a cloud platform, including scripts for the automated deployment and setup of virtual machines. This way, SmartSHARK can be installed in private clouds, to create private data repositories and analytics approaches, as well as in public clouds for the sharing of data and analytics.

The main contributions of our paper are the following.

- The SmartSHARK cloud platform for scalable software analytics based on Apache Spark.
- A software mining approach that is integrated into SmartSHARK to update data and collect information about new projects.
- The combination of different data sources that allows to combine text mining with source code, change and social metrics.

The remainder of this paper is structured as follows. In Section 2, we give an overview of the related work. Then, we describe the SmartSHARK platform in Section 3. Within that section, we describe the mining procedure and data storage, how analyses are supported, the front-end of the platform, as well as the structure of the cloud deployment. Then, we proceed with the description of a sample analytics project in Section 4, to demonstrate the power of SmartSHARK. Afterwards, we discuss the capabilities of SmartSHARK and currently known problems and limitations together with possible solutions in Section 5. Finally, we conclude and give an outlook on future work in Section 6.

2. RELATED WORK

The research on software mining and software analytics covers many different directions and aspects. However, most related work only considers software mining (e.g., [12, 18, 34, 47]) or analytic problems (e.g., defect prediction [16] or effort prediction [50]). Only very few of them consider not only mining or analytics, but actually try to combine these two aspects. Within this section, we focus only on platforms that combine the data mining with at least basic functions for analytics or already perform analytics.

Dyer et al. [24, 25] developed *Boa*, a domain-specific language and infrastructure for analyzing ultra-large-scale software repositories. It is a query system, where complicated queries can be executed to extract information from previously cached repositories using a distributed MapReduce query language. Boa programs compile down to different MapReduce jobs, which are then executed on an Apache Hadoop [6] cluster. The key difference between Boa and SmartSHARK is the type of analysis that is supported. While Boa provides Abstract Syntax Trees (ASTs) of the projects, it does not directly enable deep analytics, e.g., based on software metrics, social aspects, or similar, which are possible with SmartSHARK. Such data would have to be calculated manually for each project by the researchers, which is a great inhibition when it comes to applications such as defect prediction or effort prediction. Moreover, this would probably lead to performance problems of the analytic approaches. Furthermore, Boa heavily uses MapReduce for its queries, whereas SmartSHARK uses Apache Spark. It is reported that MapReduce is inefficient for interactive analytics [78] as they are intended to be performed with Boa. This is the main problem that should be resolved by Apache Spark.

Gousios and Spinellis [40] developed the *Alitheia Core* platform to perform "large-scale software quality evaluation studies" [40]. The architecture of Alitheia Core is divided into three different layers: (1) Result presentation, (2) system core, and (3) data mirroring, storage, and retrieval. The first layer is implemented via a web front-end. The second layer includes a job scheduler and cluster service as well as other services, which are connected via an Open Services Gateway Initiative (OSGi) interface. The third layer is responsible for the storage and retrieval of the mined data. This platform provides a metrics plug-in system, which enables researcher to implement their own plug-ins to calculate metrics out of the mined data. From a structural perspective and general idea, Alitheia Core is very similar to our approach: we also have a mining part, an analytic core, and a web front-end. However, our platform is designed around big data technologies to allow scalable analytics. Moreover, our platform is deployed in the cloud in order to allow elastic scaling of resources. Finally, our analytic core, using Apache Spark, is more powerful in terms of computational capabilities due to the usage of an Apache Hadoop cluster for job execution and provides powerful algorithms for the data analysis through Apache Spark's Mllib and GraphX libraries.

A proprietary and not publicly available approach for building a software analytics platform with integrated mining is the Microsoft internal CODEMINE [19]. CODEMINE is quite similar to SmartSHARK in its general structure: the data is collected from repositories, stored in a flexible database and exposed via a single API on which analysis and tools can be defined. However, no details on the implementation of the platform are available and we assume that it is tailored to the Microsoft internal tooling environment. In addition to a high-level description of CODEMINE, the authors also provide lessons learned for building tools similar to CODEMINE, e.g., flexibility in the analysis and the data storage, separting the storage for each project, and hosting the platform in a cloud environment. We used these lessons learned for the the design of SmartSHARK.

There are also commercial approaches that try to combine software mining with software analytics. Bitergia [13] offers several packages, which differ in the level of analysis capabilities. However, the analyses provided by Bitergia are on the level of BI, i.e., reporting of what happened in the repositories. For example, they provide dashboards that display the number of performed commits or how many authors are active in the analysed project. Similar to Bitergia is the OpenHub project [14]. OpenHub is an open platform, where every user can add or edit projects. The platform calculates different statistics for added projects, which are similar to the statistics calculated by Bitergia and also on the BI level.

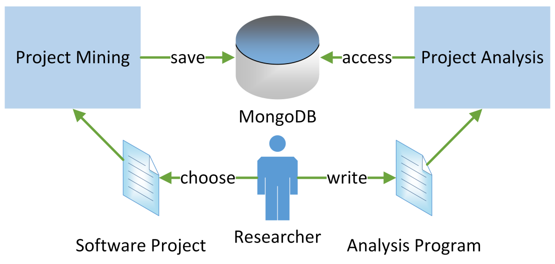


Figure 1: Design of SmartSHARK

In comparison to SmartSHARK, Bitergia and OpenHub do not support deep analytics or predictions for the future of projects. Additionally, users of Bitergia and OpenHub are not allowed to create their own analytics.

3. THE SMARTSHARK PLATFORM

Generally, the process of analysing a software project can be divided into two different steps: (1) mining of the project data and (2) analysis of the mined data. Moreover, to create a software project analytics platform that is attractive for a broad range of researchers and practitioners, the developed platform should possess the following four characteristics [15, 19]: (1) easy to use; (2) allow flexible and scalable analytics; (3) extensible with new approaches; and (4) allow analysis of different kinds of data collected from multiple sources.

SmartSHARK is designed as a general-purpose platform for supporting software analytics with the aim to fulfil all of the above characteristics, and supports both the software mining as well as the analytics. Figure 1 gives a logical overview of the platform, which is independent of the underlying infrastructure.

SmartSHARK reduces the manifold tasks of a researcher to exactly two tasks: (1) Choosing the software project, which should be analysed, and (2) the writing of an analysis program. Hence, the researcher is able to focus on the writing of the analysis program, since choosing a software project to investigate does not consume time and the mining process is automated (see Section 3.1). The results of the mining process are saved in a MongoDB. After the completion of the mining process, the researcher writes an analysis program based on Apache Spark [8, 77, 78]. This analysis program is executed on the platform and accesses the mined data in the MongoDB (see Section 3.2). All functionality is hidden behind a web front-end based on the Yii2 framework [76] (see Section 3.3). The complete SmartSHARK platform is developed as a cloud application in order to allow harnessing the scalability offered by the cloud (see Section 3.4).

3.1 Data Mining

Many tools and methods, which are used for data extraction and application, are context-specific. These tools are hard to adapt to different circumstances, because of the tight coupling between the extraction process and the application. Makedonski et al. [54] proposed a model-based software mining infrastructure to circumvent these problems. The framework relies on "homogeneous high-level domain-specific models of facts extracted from raw assets" [54]. These different domain-specific models are then combined and transformed into models, that are related to a certain assessment task. These models make use of the Eclipse Modeling Framework (EMF). The mining step in our platform is performed with the help of this model-based infrastructure, which is called DECENT. DECENT uses different tools, which are widely used in research, to extract facts from raw assets. These tools include CVSAnaly [57] for extracting information out of source code repository logs (e.g., used in [31, 46]), InFamix [48] for calculating software quality metrics (e.g., used in [4]), and DuDe [23, 74] for duplication detection. DECENT combines the results of all these tools and links them together into a single DECENT model. After the creation of the DECENT model, we transform it into an intermediate EMF model called DECENTMongo, which is used to translate the DECENT structure into the structure of the MongoDB used by SmartSHARK. Then, the DECENTMongoDBConverter tool, which is part of SmartSHARK, writes the mined data from the DECENTMongo model to the MongoDB.

As stated above, researchers only need to choose the software projects, which should be mined. The projects are added via the web front-end. SmartSHARK only requires (1) the URL of the Version Control System (VCS) and (2) the programming language of the project. As VCS, SmartSHARK currently only supports Git. The programming language is required by InFamix for the calculation of software metrics. During the mining process, the progress can be observed in the web front-end. Furthermore, each project has its own configuration files, which can optionally be edited via the web front-end to customize the mining process.

The data we obtain from the mining process are change-based. For each change (i.e. commit), we store all changed software artifacts and their location in the project. Therefore, it is possible to reconstruct the whole project structure at any point in time. The software artifacts include source code files, classes, methods, functions as well as documentation files, such as readme files.

We store five different types of data for each artifact: (1) software metrics, (2) source code changes, (3) project structure, (4) change history, and (5) bug labels. The software metrics include static source code metrics, change metrics, social metrics (e.g. developer cooperation factors). A complete list of the metrics can be found in the SmartSHARK documentation online [71]. Furthermore, we save delta values for each metric, which indicate how this metric has changed since the artifact was last touched. In addition to the metrics, we also save the concrete textual change that was made, i.e., the diff of the commit. This enables researchers to analyse the source code and the changes that were made to it. This is important for the development of new defect prediction approaches, as shown by, e.g., Tan et al. [69]. Additionally, we store bug labels and a confidence indicator label for each artifact at each change. The confidence label indicates, if the resulting bug label (artifact is bug-prone at this change or not) can be trusted. These bug and confidence labels are generated by DECENT via origin analysis [36].

3.2 Software Analytics

Out of the data we mine, we can answer questions which, for example, typically arise if we examine the evolution of a software project: How many files were changed between commit 2 and commit 70? Which files were moved in this time frame? How is the ownership of file X changing? The data can also be used for deeper analytics, like defect prediction, effort prediction, or social aspects of the software development. An analytic example is given in Section 4.

As stated above, we connected our platform directly with the Apache Spark framework for big data analytics. The core of Apache Spark is an abstraction called Resilient Distributed Datasets (RDDs). RDDs are defined as a "fault-tolerant collection of elements that can be operated on in parallel" [10]. Furthermore, they have the ability to rebuild lost data, in case of failures during job executions. Spark applications are coordinated by a SparkContext object in the main program (in Spark called: *driver program*) and run as an independent set of processes on a cluster. The SparkContext must be connected with a cluster manager. Apache Spark offers different connections or deployment methods [11]: (1) *standalone mode*, where Apache Spark itself provides a simple cluster manager; (2)*mesos mode*, where Apache Mesos [7] is used as cluster manager; and (3) *Yet Another Resource Negotiator (YARN) Mode*, where YARN is used as cluster manager. SmartSHARK uses the YARN mode of Spark. We use an Apache Hadoop cluster, where YARN runs on.

For the definition of analysis jobs, Spark supports multiple languages. With SmartSHARK we currently support only Java and Python for the definition of Spark jobs. To perform analytics on SmartSHARK, the researcher first writes their analysis program. Spark allows local testing of jobs without the need to run on the actual Hadoop cluster, which we recommend for users of SmartSHARK before job execution on the platform. Once the testing is completed, the researcher can upload and execute the Java Archive (JAR) or Python file to the platform using the web front-end. The front-end allows to add Spark arguments as well as program arguments. This allows for parametrizable analytic jobs, e.g., to define the name of the project to be analyzed or a certain threshold important for the analysis using program arguments. For the execution, the Spark jobs are submitted to the Hadoop cluster, distributed among the nodes in the cluster and processed in parallel. Information about all jobs, both running, as well as completed, are retrieved via the Representational State Transfer (REST) Application Programming Interface (API) of Hadoop and displayed in the web front-end. Furthermore, it is possible to terminate a job via the interface.

Results can be saved in the MongoDB (if the user has sufficient rights, see Section 3.3) or in the local file system. All users of the platform have their own directory, where data can be saved. Users can manage their folders with the web front-end, which allows downloading and deleting of saved files.

3.3 Web Front-End

From and end-user perspective, the web front-end is the central part of our platform. It is the only place where users directly interact with SmartSHARK and the starting point for both the mining and the software analytics. We decided to build a web based Graphical User Interface (GUI) based on the Yii2 framework [76], because of the good implementation of the Model-View-Controller (MVC) design pattern of Yii2, which allows easy extension of applications. The decision to deploy SmartSHARK online and provide users with a web front-end also resolves the problem of distributing SmartSHARK, because interested users can simply visit the home page and do not need to download and install a standalone application. The main tasks of the web front-end are the following: (1) management of mined projects, (2) software analytics, (3) user management.

Role Name	**Web Interface Permissions**	**MongoDB Permissions**
Admin	all	all
AdvancedUser	access to owned files; submission of Spark jobs; can only see fully mined projects	read, write, delete
User	same as AdvancedUser	read

Table 1: User Roles and their permissions.

3.3.1 *Management of Mined Projects*

For the project management, SmartSHARK offers the following features:

- **Add project**: Adds a project to the project list, which can be mined later.
- **List projects**: Shows all projects, together with their mining progress. Furthermore, sorting and searching functionalities are implemented.
- **View project**: Displays detailed information about the project, like a graph depicting the artifacts that were changed in each commit.
- **Start mining**: Starts the mining process for the chosen project.
- **Delete project**: Deletes the project and all its data.

3.3.2 *Software Analytics Front-end*

The analysis feature comprises of the submission and monitoring of jobs, as described in Section 3.2, as well as the results management. Users can download and delete the results stored in their private folder via the web interface. Depending on the user rights, the submitted jobs may be allowed to modify the underlying MongoDB with the mined data. Moreover, the front-end contains examples for analytic jobs which can be used as a cookbook for the definition of new analytic jobs.

3.3.3 *User Management*

SmartSHARK implements Create, Read, Update, Delete (CRUD) user management functionalities. Each SmartSHARK user has a role that is assigned via the web front-end. The role specifies the permissions for the web front-end and the MongoDB access. Table 1 shows the different roles and their permissions. The three defined roles allow the separation between users who are allowed to trigger the mining of projects and adapt the configuration of the platform (Admin role), users that are allowed to create analytics that modify the MongoDB (AdvancedUser role) and users that can only create analytics that access the MongoDB without write privileges (User role). This can be useful if students should work with the platform to make sure that they cannot delete or overwrite data by mistake.

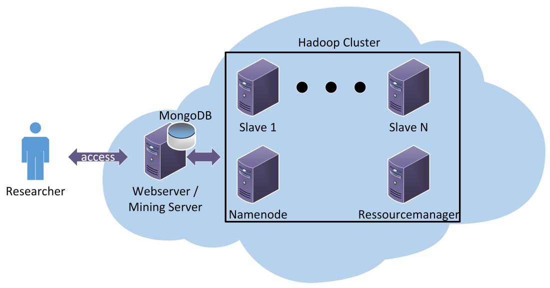


Figure 2: Infrastructure of SmartSHARK.

3.3.4 Additional Features

The web front-end has several features, which are important for conducting research. First, the web interface provides a backup functionality for the project data. The researcher needs to provide a backup name and select which collections of the MongoDB should be backed up. It is possible to, e.g., backup all the mined data, or only the data about the methods of the projects. This feature is especially useful for the preparation of a publication, where a fixed data set is required which may not be changed afterwards. SmartSHARK allows the download of created backups, as well as the import of backups into its database. However, the import does not integrate with the existing data, but overwrites the complete mined data.

Moreover, the web front-end was extended to show some analysis results for each project directly, instead of having the results only available as download. This was performed as part of the analytics example we provided to show how SmartSHARK can be modified to directly put results into operation and make them actionable.

3.4 Cloud Deployment

SmartSHARK is developed as a cloud platform. Currently, we are running SmartSHARK in a small private cloud that we locally operate within our research group for research purposes. The infrastructure of our SmartSHARK deployment is depicted in Figure 2. Currently, we have a web-server and an Apache Hadoop cluster. Table 2 shows the Virtual Machines (VMs) and their specifications. The webserver needs to have a large amounts of RAM, it currently fulfills mutliple roles: it does not only host the web front-end, but is also responsible for the mining of projects and the hosting of the MongoDB. The RAM is mostly required for the mining, because the EMF models created during the mining must fit completely into the RAM. We are aware that this is not ideal, but currently cannot deploy SmartSHARK in a better way, with dedicated servers for the MongoDB and the mining, due to a limitation of resources. The Apache Hadoop cluster executes analytics jobs defined with Apache Spark.

Our aim was to develop SmartSHARK in a way, that the deployment is as easy as possible and the deployment model is adaptive, to allow changes based on the available computational resources, e.g., using dedicated servers for the MongoDB and the mining. Therefore, we use DevOps technologies to provide scripts that fully automate. The deployment process is twofold: First, VMs have to be created and provisioned in our cloud. Second, the created machines need to be configured.

Hostname	Virtual CPU	RAM	Disk Space
namenode	2 / 64bit	4GB	40GB
resource-manager	2 / 64bit	4GB	40GB
slave1	2 / 64bit	4GB	40GB
slave2	2 / 64bit	4GB	40GB
webserver	8/ 64bit	16GB	160GB

Table 2: VMs of the SmartSHARK Deployment.

3.4.1 Creation of VMs

We use Vagrant [43] for the creation and provisioning of VMs. Vagrant allows the definition of scripts that specify which and how many VMs shall be created. The first part of the deployment is adding the user credentials for the cloud to the script. The second part is the specification of the required VMs. This is done using so-called flavors. A flavor defines the available resources of a VM, i.e., the computational capabilities in terms of CPUs, RAM, as well as disk space. Within the deployment script, it is specified how many VMs of which flavors are required. The Vagrant script also defines how many slave nodes the Hadoop cluster has.

Once the VMs are created, they must be provisioned, i.e., instantiated with an operating system. Vagrant allows the selection of predefined images of operating systems which can be downloaded for this purpose. For SmartSHARK, we are currently using Ubuntu 14.04. Once the provisioning is performed, we have the required VMs up and running with an operating software, but no other software installed yet.

3.4.2 VM Configuration

We use Ansible [5] for the configuration of the VMs. Ansible can be called using Vagrant and allows the definition of so-called playbooks, i.e., scripts that define which software should be installed in a VM and how the software should be configured. To achieve this, Ansible allows calling bash commands required for the setup of applications. This approach, compared to manually configuring the systems, has several advantages: First, it makes versioning possible. Hence, the infrastructure can be recreated at any point in time. Furthermore, the system configuration can be split into smaller pieces, which are better manageable. These aspects are important for the extensibility of SmartSHARK as well as for its deployment on different infrastructures.

The configuration of SmartSHARK requires six different playbooks. Each playbook serves a different role for the configuration of SmartSHARKs infrastructure. The playbooks are the following.

- **common**: Installs base software, which is needed on every VM, e.g., Java, Network File System (NFS).
- **hadoop_common**: Downloads Hadoop and creates the Hadoop user on every VM.
- **hadoop_configuration**: Configures Apache Hadoop on every node. It also makes the Hadoop Distributed File System (HDFS) available via a POSIX compatible file system using an NFS wrapper.
- **services**: Installs required services on every node; the services depend on the node type.
- **webserver_setup**: The setup for the webserver. It contains tasks, like the installation of Apache2, cre-

Software	Version
Apache Hadoop	2.6.0
Ansible	1.9.2
Vagrant	1.7.2
Java	OpenJDK 64-bit 1.7.0_79
Linux	Ubuntu Server 14.04
MySQL	5.5.43
MongoDB	3.0.4
Apache2	2.4.7
PHP	5.5.9
Apache Spark	1.4.0
Yii2	2.0.5

Table 3: Software the SmartSHARK Deployment.

ation of the configuration for the web front-end, installation of required software (e.g., git, PHP5, curl, MongoDB, MySQL, Apache Spark), installation of plugins (e.g. PHP-MongoDB extension) and execution of a Yii2 migration. The purpose of the Yii2 migration is the instantiation of the MongoDB user collection with default values.

- **format_hdfs**: Formats the HDFS and is, therefore, only executed on the name node.

For the configuration of the VMs the software versions listed in Table 3 are used. Because we use a scripted deployment method, we only require adaption of the Ansible playbooks to change the version of one of the used software versions and/or add additional software.

3.4.3 Requirements on the Cloud Environment

In order to be able to deploy SmartSHARK, the cloud environment must support Vagrant. Ansible requires only SSH and can, therefore, be used with any cloud provider. To run the deployment, only two bash-commands must be called. `vagrant up -no-provision` for the creation of the VMs and `vagrant provision` for their configuration.

To deploy SmartSHARK on other cloud environments that support Vagrant, the scripts may have to be adapted since parts of the deployment are cloud provider specific, e.g., the flavors of the VMs and the user credentials.

4. SOFTWARE ANALYTICS WITH SMARTSHARK

In order to demonstrate the power of SmartSHARK, we performed some experiments with it. First, we evaluated the mining part of the platform, by cloning multiple projects and collecting their data. Then, we created a sample analysis in which we show how defect prediction models can be created for the previously mined projects. Moreover, we adapted the SmartSHARK front-end to directly show the results of our sample analysis in order to allow quick feedback to researchers and developers interested in such analysis and to provide a major step towards making the analysis actionable.

4.1 Data Collection

To evaluate if the mining of projects works as desired, we selected some projects from GitHub [35]. We did not follow any specific methodology for the selection of projects, instead we just selected 20 projects randomly by browsing

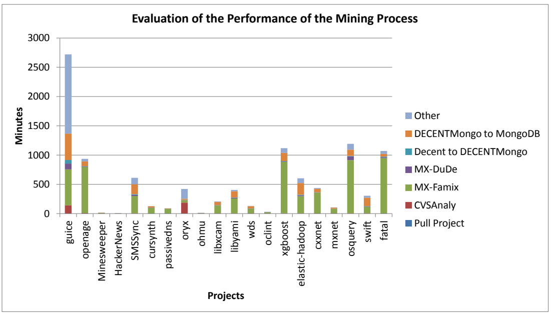


Figure 3: Performance of the Mining Process.

projects with the *explore* function of GitHub. The only requirement for a project was that they were programmed in Java, C or C++. Table 4 lists the mined projects, including the number of commits, the number of files in the repository, the size of the repository, the programming language, and a very brief project description. The number of files and the size are reported for the latest revision of the repository. For each of the selected projects, we triggered the mining via the web front-end of SmartSHARK. We measured how long the different steps of the mining process took and recorded if we encountered problems.

Figure 3 shows the time consumed for the mining. The differences in the overall runtime between the projects are a result of the differences in project size, number of files, and number of commits. Even though the mining is only running on a single core of the Web server, the largest project only took less than 45 hours to mine. Most time is consumed by the collection of the metric data with InFamix and the DECENTMongoDBConverter, i.e., the component that stores the mined data in the MongoDB. There are two outliers, where the runtime is consumed differently. The first is guice, where the addition of the diffs consumes most of the runtime (included in other). The second outlier is the oryx project, for which the analysis of the repository itself consumed the most time. When we investigated this, we determined that CVSAnaly stopped working for a while, but then simply resumed as if nothing happened. Upon further investigation we determined that this was an issue with CVSAnaly due to a race condition in the implementation.

For Mahout [9], the mining simply stopped due to a failure of InFamix. InFamix did not throw any errors, it just stopped working. Therefore, the process did not finish. We were not able to track down the problem. But besides this incident, there were no other failures in mining for the projects that we have tested.

4.2 Software Analytics Example

In order to demonstrate the analytic capabilities of SmartSHARK, we implemented some examples for software defect prediction. The example is not designed to have the highest performance possible or implement a very complex defect prediction scheme. Instead, we want to demonstrate how the data offered by SmartSHARK can be used and combined for an analytic goal. Defect prediction is used to identify defect-prone entities (e.g. source code files) in advance. Typically, defect prediction models use a training set, which contains various measures (e.g. code metrics) of available entities and the history of the entities defect proneness. The prediction

Project	Lang.	#Commits	Size	#Files	Description
guice [38]	Java	1442	89 MB	713	Dependency injection framework for Java.
openage [65]	C++	1761	8 MB	560	Project, which clones Age of Empires II.
Minesweeper [53]	Java	65	7 MB	207	Minesweeper game for Android.
HackerNews [52]	Java	12	1 MB	78	Pulls top stories from the HackerNews API for Android.
SMSSync [73]	Java	1395	40 MB	557	SMS gateway for Android.
cursynth [72]	C++	219	3 MB	185	MIDI enabled, subtractive synthesizer, for the terminal.
passivedns [32]	C	220	1 MB	50	Network sniffer, which logs DNS server replies.
oryx [17]	Java	372	48 MB	537	A real-time large-scale machine learning infrastructure.
ohmu [39]	C++	226	3 MB	185	Compiler intermediate language for static analysis.
libxcam [1]	C++	250	3 MB	242	Project, with the aim to extend camera features.
libyami [2]	C	487	4 MB	248	Library for media solutions on Linux.
wds [3]	C++	238	3 MB	193	Libraries for building Miracast/WiDi-enabled applications.
oclint [63]	C++	733	3 MB	349	Static source code analysis tool.
xgboost [22]	C++	1847	8 MB	373	Large-scale, distributed gradient boosting library.
elasticsearch-hadoop [26]	Java	1243	9 MB	576	Elasticsearch real-time search and analytics natively integrated with Hadoop.
cxxnet [20]	C++	852	4 MB	173	Concise, distributed deep learning framework.
mxnet [21]	C++	236	1 MB	124	Combines ideas from projects for machine learning.
osquery [29]	C++	2208	9 MB	555	Operating system instrumentation framework.
swift [30]	Java	496	8 MB	427	Annotation-based Java library for Thrift.
fatal [28]	C++	401	3 MB	141	Library for fast prototyping of software in C++.

Table 4: Information about the Mined Projects.

model is fit to the training set. Afterwards, the model is used to predict defects of entities in the future. This preselection of entities, which most likely contain a bug, is done, as budget, people and time, are often limited in development organizations and therefore only some of the code can be tested before releasing it [61].

4.2.1 Training and Test Data

The first decision one has to make is how to create training and test data for such an approach. We are using a within-project defect prediction approach, i.e., we built the defect prediction model on the past of a project and evaluate it on later revisions of the same project. Data from other projects are not used during the training of a prediction model. To select the training and test data within a project, we took a pattern from [69]: We leave a short gap in the beginning, i.e., we are excluding the first commits of a project, because the change patterns may not be stable in the beginning [44, 51]. Then, we take the next commits of the data for training, leave a gap, and use the next period of the project to evaluate the prediction model. Moreover, we do not consider all commits until the end of the project, but leave another gap, because here the data is usually worse in terms of bug information, because there was less time for the identification of the bugs. We use the information from the origin analysis of DECENT based on the bug labels of CVSanaly to identify which commits contained bugs. This information is already contained in the data through the mining.

In case the project lifetime is shorter than 1400 commits, we did not exclude the first commits. Additionally, we excluded the project "HackerNews" from our analyis due to the short history of only 12 commits.

4.2.2 Classification Models

We created three different classification models for each project. Each of the models uses different data and shows different possibilities of SmartSHARK.

The first classification model is based only on the social, change, and static source code metrics. Using the metrics as input, we trained a random forest for the classification. The random forest is one of the machine learning algorithms available in Spark's Mllib. Training defect prediction based on software metrics is a commonly used method in research [16]. Due to the powerful mining based on DECENT the combination of social aspects, change metrics, and static source code metrics does not pose any problems with SmartSHARK. In the literature, we know only one example where all three kinds of metrics were combined, however, this was not in the context of defect prediction, but vulnerability prediction [67].

The second classification model is based only on the *diff* characteristic for text classification. *Diff* is calculated by the differences between the last state and the new state of the source code of the artifact. We created a bag-of-words for the diffs and used the tf-idf [62] metric to estimate the impact of certain terms. Based on this, we created a naïve bayes classifier for the classification into defect-prone and non-defect-prone files. The naïve bayes classifier is also available in Spark's Mllib.

The third classification model is a combination of the two approaches explained above, in order to demonstrate that SmartSHARK allows the combined analysis based on software metrics and text analysis. The third model internally trains three prediction models. The first two are the random forest and the naïve bayes model as described above. Additionally, we train a logistic regression model (also available in Spark's Mllib) based on the software metrics, i.e., on the same data as the random forest. Then we use an ensemble learning method called bagging [75] to determine the overall classification. The random forest, the naïve bayes classifier and the logistic regression model separately classify an instance. Then, we use the majority vote of those three internal models to determine the overall classification of the third model.

The results of our defect predictions are written back to the MongoDB as new collection that is stored together with the analyzed entity. Through storing the analysis results in the MongoDB, we enrich the database with more information that was not gained through the mining directly, but rather with advanced software analytics based on SmartSHARK itself. Therefore, SmartSHARK can feed itself with more data.

4.2.3 Prediction Results

We store the confusion matrices for the classifiers in the file system, from where users can download them in order to analyze the prediction performance. Moreover, we modified the SmartSHARK front-end to be able to visualize prediction results and output performance metrics for defect predictions. We decided to perform this adaptation to show that a platform like SmartSHARK also provides a way towards actionable software analytics, as the results of predictions can be integrated into the front-end. Hence, our front-end displays information about the project history similar to Bitergia and OpenHub, but also supports the visualization of the results of deep software analytics.

We implemented the following front-end features for showing and evaluating the prediction results.

- Zoomable graphs that depict the regions of the project used for training and testing. Furthermore, these graphs show how many software artifacts are actually defect-prone according to the data, and how many are predicted as defect-prone, as well as the differences between the prediction and the actual values.

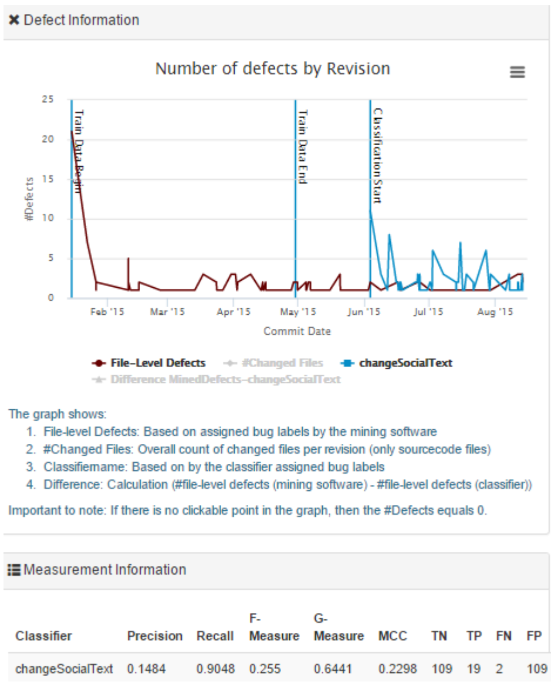
- Visualization of the so-called confidence cut, i.e., the line after which we are not sure if the bug labels in the data are correct, due to a small number of commits afterwards. Tan et al. [69] suggest to choose this such that the time from the last commit till the cut is the average bug fixing time. As this data is not yet available in SmartSHARK, we chose a time span of three years based on findings by Jiang et al. [49]. Additionally, SmartSHARK displays what data was used for training, the classifier, and from which commit the classification starts.

- The confusion matrix of a classifier as well as multiple performance metrics, including the prediction error, precision, recall, F-measure, G-measure, and Matthews Correlation Coefficient (MCC).

In Figure 4, we show a screenshot of the web front-end that shows the results for the third classifier, i.e., the one that combines software metrics and text analysis for the project passivedns. We do not provide a detailed analysis of the performance of the different classification models within this paper, because this is out of scope. Our aim was not to create a high performance defect prediction model, for which such an analysis is warranted and necessary, but rather to show the analytic capabilities of SmartSHARK.

5. DISCUSSION

Within this section, we want to discuss the capabilities of SmartSHARK, currently known limitations, and possible solutions. We separate this discussion into four parts, analogous to the structure of SmartSHARK, i.e., the data mining, the software analytics, the web front-end, and the cloud deployment.



The graph shows:

1. File-level Defects: Based on assigned bug labels by the mining software
2. #Changed Files: Overall count of changed files per revision (only sourcecode files)
3. Classifiername: Based on by the classifier assigned bug labels
4. Difference: Calculation (#file-level defects (mining software) - #file-level defects (classifier))

Important to note: If there is no clickable point in the graph, then the #Defects equals 0.

Measurement Information

Classifier	Precision	Recall	F-Measure	G-Measure	MCC	TN	TP	FN	FP
changeSocialText	0.1484	0.9048	0.255	0.6441	0.2298	109	19	2	109

Figure 4: Results in the Web Front-end

5.1 Data Mining

The data mining is an important and powerful feature of SmartSHARK. Since it is responsible for the collection of the analysis data, it is a vital part of SmartSHARK. As we showed, we can collect the data of many projects without any problems and only experienced one failure of the mining (see Section 4.1). The model-based mining process based on DECENT provides a certain independence of the concrete tools used to collect the data. E.g., to substitute a tool, we only have to adapt the extraction of the DECENT facts from the raw assets created by the tool [54]. All other aspects of the mining remain untouched. Similarly, we can easily add new information sources to SmartSHARK. We just have to extend the DECENT meta-model with additional attributes that fit the structure of the new facts. For example, if want to mine mailing lists, we first extract the relevant facts with a third party tool, e.g., MailingListStats [58]. Then, we map the facts to a corresponding meta-model and provide a model-to-model transformation, e.g., by means of Epsilon Transformation Language (ETL) or Epsilon Object Language (EOL), which integrates the mined facts into the DECENT model and we are finished. The rest of the mining process remains untouched.

However, the data mining is not without problems and limitations. Table 5 summarizes the current limitations of SmartSHARKS mining process. The limitations (1)-(4) are problems of the mining process itself. The execution of InFamix is sometimes unreliable, as our problem with the mining of Mahout demonstrates (see Section 4.1). We need to further analyze the source of this problem. Depending on the severity and frequency of this mining issue, we may have to replace InFamix with another metric calculation tool. More-

Data Mining	
Limitation	**Possible Solution**
(1) InFamix unreliable	Substitute InFamix
(2) Multiple programming languages within a project not yet supported	Substitute InFamix or execute InFamix multiple programming languages
(3) Bug labels are assigned heuristically	Integrate Issue Tracking System facts
(4) No incremental update of projects	Adapt mining process
(5) InFamix slow	Massive parallelization
(6) Whole EMF model must fit into the RAM during mining	Provide more resources or implement EMF-Fragments [64]
(7) DECENTMongoDBConverter slow due to double pass to set cross references	Refactor the DECENT-MongoDBConverter to use Apache Spark
(8) Only one project can be mined at a time	Provide more resources

Table 5: Limitations of the Data Mining.

over, InFamix can only calculate software metrics for one project at a time. Since we currently only execute InFamix once, this also means that we currently can also only calculate software metrics for one programming language within SmartSHARK. The solution for this is to extend the configuration possibilities of SmartSHARK and allow the selection of multiple programming languages. We can then execute InFamix once for each programming language. However, InFamix only supports Java, C, and C++ and we, therefore, might need to consider other tools for different languages. The third problem is related to the current identification of the bug labels, based on the *BugFixMessage* extension of CVSAnaly. CVSAnaly marks revisions as bugfix revisions by looking in the commit message and searching for keywords like *bug* or *fix*. This information is sometimes error-prone or missing (e.g., a developer fixed a bug but does not mention it in the commit message). A partial solution for this problem would be the integration of facts extracted from an issue tracking system, e.g., with BZExtractor, as proposed by Makedonski et al. [54], to improve the bug label assignment. Moreover, we currently cannot update the mined data of projects, because of limitations in the model-based mining framework. A workaround is to delete a project and re-collect the data at a later point. However, this does not scale well, especially with larger projects. Instead, the mining process should be adapted to allow incremental mining.

The problems (5)–(6) are two bottlenecks regarding the duration of the mining process (see Section 4.1). The first bottleneck is the very slow execution of InFamix, because every version of a project is analyzed subsequently. We see two possible solutions to this problem: either we add more resources to our mining process that allow the execution of InFamix in parallel, e.g., by also using the Hadoop cluster for this task or provide a separate cluster for InFamix. Moreover, we could try to incorporate a modern analysis method like the LISA parser [4] for the fast analysis of the complete history of a project. The second bottleneck in the mining process is the conversion of the DECENT model to the MongoDB. However, our implementation of the DECENT-MongoDBConverter leaves plenty of room for optimization. For example, we could use Apache Spark internally to speed up the process through massive parallelization.

The seventh limitation we found for the mining is due to the model-based mining approach. All models must fit into the RAM during the mining. This limitation could be overcome by splitting the model into fragments [64].

The limitations (5)–(7) are all limitations due the available resources. Our eigth limitation is a corollary to this: we can currently mine only one project at a time. However, in case we would have multiple VMs available just for the mining of projects, we could easily process multiple projects at a time.

5.2 Software Analytics

As the analytics examples in Section 4.2 show, the analytics based on the mined data in the MongoDB with Apache Spark allows flexible and scalable analysis jobs. Java and Python are popular languages which should facilitate an easy adoption of Apache Spark and, by extension, SmartSHARK. Moreover, Apache Spark provides good support for the connection with the MongoDB, e.g., through the deep framework [68]. Moreover, there are already libraries for Apache Spark which are useful for the analysis of software projects, e.g., the *Mllib* for machine learning which we also use in our analytic examples, or *GraphX* for graph and graph-parallel computations.

In our examples, we used only defect prediction and only data of one project. However, there is no reason why data from multiple projects cannot be used for the analytics, or why other interesting topics, like effort prediction or software evolution in general could not be studied.

We currently see only one major limitation to the analytics. The Mllib of Apache Spark is rather small in comparison to the possibilities offered by languages like R [70] or tools like Weka [42]. This limitation can only be overcome by the extension of the existing or the creation of new libraries for software analytics with Apache Spark.

5.3 Web Front-end

The web front-end provides an easy-to-use starting point for the analysis of projects. It provides a convenient interface for administrators to add projects with the integrated data mining, as well as job monitoring for the currently running Spark jobs. For researchers, it facilitates the submission of own analysis jobs as well as the download of all created results. Through the provision of examples on the front-end, we provide a cookbook style guideline for new analyses. The separation of the users in three different roles provides basic security functions, e.g., by restricting who is allowed to write to the MongoDB.

We currently see one limitation for our web front-end. The creation of the zoomable graphs for the projects can become quite large due to the potentially very high number of commits in projects. This can significantly slow down the loading time of pages. We could solve this either by precalculating the graphs and, thereby, speed-up the loading time at the cost of more resources at the server side or try to use advanced visualization techniques that require less resources to depict the same amount of information.

5.4 Cloud Deployment

Our cloud deployment of SmartSHARK allows the scaling of the platform, which makes it suitable for analysing

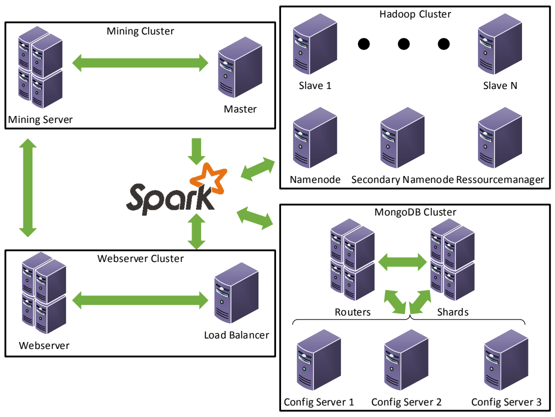


Figure 5: Optimized SmartSHARK Deployment

huge amounts of data, as well as executing highly resource-demanding analytics. Moreover, since the platform is readily available for researchers, they can concentrate on writing analytics programs and do not have to setup and maintain the analysis infrastructure. Due to our usage of automated deployment scripts, we facilitate that SmartSHARK can be easily adapted to both: new underlying software as well as a new cloud infrastructure. This also facilitates the recovery of SmartSHARK in case of fatal problems.

However, our current deployment is responsible for many of the above listed limitations to the mining and analytics, as they are due to the lack of resources which tied our hands to some degree. Therefore, we would like to port SmartSHARK to a larger cloud infrastructure in the future. Figure 5 illustrates, how an optimized large-scale cloud deployment of SmartSHARK can look like. We replace the single VM for the webserver with a webserver cluster with one or more load balancers. Furthermore, instead of performing the mining on the webserver, we deploy a separate mining cluster. This mining cluster has a master node, which distributes the different mining tasks to the mining nodes. Additionally, this mining cluster communicates with the Apache Hadoop cluster via Apache Spark e.g., for the execution of the DECENTMongoDBConverter.

The Apache Hadoop cluster consists of the namenode and the secondary namenode, as well as the resourcemanager and a specified amount of slaves. The webserver can communicate via Apache Spark with the Apache Hadoop cluster. Furthermore, instead of running the MongoDB on the webserver, we introduce the MongoDB cluster. The MongoDB cluster uses sharding [60] to make read and write accesses more efficient. To this aim, we deploy shards to store the data, routers to processes target operations and queries to shards and three configuration servers to store the cluster metadata. The MongoDB documentation highlights that exactly three configuration servers are required [60].

Figure 5 does not show, that all these clusters are deployed in a cloud environment, but this is strongly recommended as the addition of new resources to the machines is easier this way. Furthermore, the communication flow in the Apache Hadoop cluster is not depicted.

This optimized infrastructure could be easily set up via Vagrant and Ansible by adapting our scripts. The reason, why we have not yet deployed SmartSHARK this way is our current limitation on resources.

6. CONCLUSION

We presented SmartSHARK, a cloud platform that combines software mining with software analytics. The software mining is model-based, which allows easy integration of new data into the mining process as well as substitution of used tools. All mined data is stored in a MongoDB, i.e., a state-of-the-art NoSQL database that can easily hold terabytes of data. For the definition of analysis jobs, SmartSHARK uses Apache Spark with an Apache Hadoop cluster in the back-end for the parallel and efficient execution of tasks. Users can access SmartSHARK via a web front-end, through which they can submit analysis jobs and download the results. Using an analytic example, we showed how SmartSHARK can be used to define powerful software analytics. Our example demonstrates that the Mllib of Apache Spark provides a good foundation for deep analytics with machine learning and that different kinds of information available from the mining, i.e., metric data and textual differences can be combined without much effort.

In the future, we plan to migrate SmartSHARK to a larger cloud environment, where we can achieve a deployment closer to the optimized deployment described in Section 5.4. Moreover, we plan to enhance the mining process by incorporating more tools, e.g., BZExtractor [54] to include data from Bugzilla or MailingListStats [58] to incorporate mailing lists.

Furthermore, we want to build libraries based on Apache Spark that are useful for software analytics, e.g., for benchmarking, analysis across multiple projects, or analysis of social and dependency graphs. Through the provision of such libraries, we want to contribute to a solution of the problem of non-replicable studies, which remains till today [55]. The usage of different tooling and platforms for mining and analysing the data is one of many reasons for this. Non-replicable studies force researcher to start over instead of reusing and improving results from a colleague [12]. Furthermore, meta-analysis (analyse an analysis) are not possible because of this problem. Through the provision of a common foundation, SmartSHARK can at least partially solve this issue.

Acknowledgements

The authors would like to thank Fabian Glaser, Michael Göttsche, and Gunnar Krull for their support for regarding the cloud technologies and deployment.

7. REFERENCES

[1] 01org. Libxcam GitHub. `https://github.com/01org/libxcam`. [accessed 28-August-2015].

[2] 01org. Libyami GitHub. `https://github.com/01org/libyami`. [accessed 28-August-2015].

[3] 01org. Wds GitHub. `https://github.com/01org/wds`. [accessed 28-August-2015].

[4] C. V. Alexandru and H. C. Gall. Rapid Multi-Purpose, Multi-Commit Code Analysis. In *Proc. IEEE/ACM 37th Int. Conf. on Softw. Eng. (ICSE)*, pages 635–638. IEEE/ACM, 2015.

[5] Ansible Inc. Ansible Documentation. `http://www.ansible.com/`. [accessed 28-August-2015].

[6] Apache Software Foundation. Apache Hadoop. `https://hadoop.apache.org/`. [accessed 28-August-2015].

[7] Apache Software Foundation. Apache Mesos. `https://mesos.apache.org/`. [accessed 28-August-2015].

[8] Apache Software Foundation. Apache Spark. `https://spark.apache.org/`. [accessed 28-August-2015].

[9] Apache Software Foundation. Mahout GitHub. `https://github.com/apache/mahout`. [accessed 28-August-2015].

[10] Apache Software Foundation. Spark 1.4 Documentation. `https://spark.apache.org/docs/latest/programming-guide.html`. [accessed 28-August-2015].

[11] Apache Software Foundation. Spark Cluster Mode Overview. `https://spark.apache.org/docs/latest/cluster-overview.html`. [accessed 28-August-2015].

[12] J. Bevan, E. J. Whitehead Jr, S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *ACM SIGSOFT Softw. Eng. Notes*, volume 30, pages 177–186. ACM, 2005.

[13] Bitergia. Bitergia. `http://bitergia.com/`. [accessed 28-August-2015].

[14] Black Duck Software, Inc. Open HUB. `https://www.openhub.net/`. [accessed 28-August-2015].

[15] R. P. L. Buse and T. Zimmermann. Information Needs for Software Development Analytics. In *Proc. 34th Int. Conf. on Softw. Eng. (ICSE)*, 2012.

[16] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[17] Cloudera. Oryx GitHub. `https://github.com/cloudera/oryx`. [accessed 28-August-2015].

[18] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Softw. Eng.*, 31(6):446–465, 2005.

[19] J. Czerwonka, N. Nagappan, and W. Schulte. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *IEEE Softw.*, 30(4):64–71, 2013.

[20] Distributed Machine Learning Common. Cxxnet GitHub. `https://github.com/dmlc/cxxnet`. [accessed 28-August-2015].

[21] Distributed Machine Learning Common. Mxnet GitHub. `https://github.com/dmlc/mxnet`. [accessed 28-August-2015].

[22] Distributed Machine Learning Common. Xgboost GitHub. `https://github.com/dmlc/xgboost`. [accessed 28-August-2015].

[23] U. Draisbach and F. Naumann. Dude: The duplicate detection toolkit. In *Proc. Int. Workshop on Quality in Databases (QDB)*, 2010.

[24] R. Dyer, H. A. Nguyen, H. Rajan, and T. Nguyen. Boa: Ultra-Large-Scale Software Repository and Source Code Mining. *ACM Trans. Softw. Eng. and Methodology*, forthcoming, 2015.

[25] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Proc. IEEE/ACM 35th Int. Conf. on Softw. Eng.*, 2013.

[26] Elasticsearch BV. Elasticsearch-hadoop GitHub. `https://github.com/elastic/elasticsearch-hadoop`. [accessed 28-August-2015].

[27] EMC Education Services. *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*. John Wiley & Sons, 2015.

[28] Facebook Inc. Fatal GitHub. `https://github.com/facebook/fatal`. [accessed 28-August-2015].

[29] Facebook Inc. Osquery GitHub. `https://github.com/facebook/osquery`. [accessed 28-August-2015].

[30] Facebook Inc. Swift GitHub. `https://github.com/facebook/swift`. [accessed 28-August-2015].

[31] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens. What does it take to develop a million lines of open source code? In *Open Source Ecosystems: Diverse Communities Interacting*, pages 170–184. Springer, 2009.

[32] E. Fjellskål. Passivedbs GitHub. `https://github.com/gamelinux/passivedns`. [accessed 28-August-2015].

[33] Free Software Foundation. GNU Diffutils. `http://www.gnu.org/software/diffutils/`. [accessed 28-August-2015].

[34] D. M. German. Mining CVS repositories, the softChange experience. *Evolution*, 245(5,402):92–688, 2004.

[35] I. GitHub. GitHub. `https://github.com/`.

[36] M. Godfrey and Q. Tu. Tracking structural evolution using origin analysis. In *Proc. Int. Workshop on Principles of Softw. Evolution (IWPSE)*, 2002.

[37] J. M. González-Barahona and G. Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empir. Softw. Eng.*, 17(1-2):75–89, 2012.

[38] Google. Guice GitHub. `https://github.com/google/guice`. [accessed 28-August-2015].

[39] Google. Ohmu GitHub. `https://github.com/google/ohmu`. [accessed 28-August-2015].

[40] G. Gousios and D. Spinellis. Alitheia core: An extensible software quality monitoring platform. In *Proc. 31st Int. Conf. on Softw. Eng.*, 2009.

[41] I. Grigorik. GitHub Archive. `https://www.githubarchive.org/`. [accessed 28-August-2015].

[42] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[43] HashiCorp. Vagrant. `https://www.vagrantup.com/`.

[accessed 28-August-2015].

[44] A. E. Hassan and R. C. Holt. The top ten list: Dynamic fault prediction. In *Proc. 21st IEEE Int. Conf. on Softw. Maint. (ICSM)*, pages 263–272. IEEE, 2005.

[45] S. Herbold. Crosspare: A tool for benchmarking cross-project defect predictions. In *Proc. 4th Int. Workshop on Software Mining (SoftMine)*, 2015.

[46] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Forecasting the number of changes in Eclipse using time series analysis. In *Proc. 4th Int. Workshop on Mining Softw. Repositories (MSR)*, 2007.

[47] J. Howison, M. S. Conklin, and K. Crowston. Ossmole: A collaborative repository for floss research data and analyses. In *Proc. 1st Int. Conf. on Open Source Softw.*, 2005.

[48] Intooitus. InFamix. `https://www.intooitus.com/company/news/introducing-infamix-free-ccjava-parser-moose`. [accessed 28-August-2015].

[49] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In *Proc. IEEE/ACM 28th Int. Conf. on Automated Softw. Eng. (ASE)*, 2013.

[50] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Trans. Softw. Eng.*, 33(1):33–53, Jan 2007.

[51] S. Kim, E. J. Whitehead Jr, and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Trans. Softw. Eng.*, 34(2):181–196, 2008.

[52] E. Lawlor. HackerNews GitHub. `https://github.com/lawloretienne/HackerNews`. [accessed 28-August-2015].

[53] E. Lawlor. Minesweeper GitHub. `https://github.com/lawloretienne/Minesweeper`. [accessed 28-August-2015].

[54] P. Makedonski, F. Sudau, and J. Grabowski. Towards a model-based software mining infrastructure. *ACM SIGSOFT Softw. Eng. Notes*, 40(1):1–8, 2015.

[55] T. Mende. Replication of defect prediction studies: problems, pitfalls and recommendations. In *Proc. 6th Int. Conf. on Predictive Models in Softw. Eng.*, 2010.

[56] T. Menzies, M. Rees-Jones, R. Krishna, and C. Pape. The promise repository of empirical software engineering data. `http://openscience.us/repo`. North Carolina State University, Department of Computer Science [accessed 28-August-2015].

[57] Metrics Grimoire. CVSAnaly GitHub. `http://github.com/MetricsGrimoire/CVSAnalY`. [accessed 28-August-2015].

[58] Metrics Grimoire. MalingListStats GitHub. `https://github.com/MetricsGrimoire/MailingListStats`. [accessed 28-August-2015].

[59] MongoDB Inc. MongoDB. `https://www.mongodb.org/`. [accessed 28-August-2015].

[60] MongoDB Inc. MongoDB Sharding Introduction. `http://docs.mongodb.org/manual/core/sharding-introduction/`. [accessed 28-August-2015].

[61] F. Rahman, D. Posnett, and P. Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proc. ACM SIGSOFT 20th Int. Symp. on the Found. of Softw. Eng.*, 2012.

[62] J. Ramos. Using tf-idf to determine word relevance in document queries. In *Proc. first instructional Conf. on machine learning*, 2003.

[63] R. Saito. Oclint GitHub. `https://github.com/oclint/oclint`. [accessed 28-August-2015].

[64] M. Scheidgen. Reference representation techniques for large models. In *Proc. Workshop on Scalability in Model Driven Eng.*, 2013.

[65] SFTtech. OpenAge GitHub. `https://github.com/SFTtech/openage`. [accessed 28-August-2015].

[66] W. Shang, B. Adams, and A. E. Hassan. An experience report on scaling tools for mining software repositories using mapreduce. In *Proc. IEEE/ACM Int. Conf. on Automated Softw. Eng.*, 2010.

[67] Y. Shin, A. Meneelay, L. William, and J. A. Osborne. Evaluating complexity, code churn, and developer activity as indicators of software vulnerabilities. *IEEE Trans. Softw. Eng.*, 37(6):772–787, 2011.

[68] Stratio. Deep Spark Framework GitHub. `https://github.com/Stratio/deep-spark`. [accessed 28-August-2015].

[69] M. Tan, L. Tan, S. Dara, and C. Mayeux. Online Defect Prediction for Imbalanced Data. In *Proc. IEEE/ACM 37th Int. Conf. on Softw. Eng. (ICSE)*, 2015.

[70] The R Foundation. The R Project for Statistical Computing. `https://www.r-project.org/`. [accessed 28-August-2015].

[71] F. Trautsch. SmartSHARK MongoDB Design. `http://smartshark.informatik.uni-goettingen.de/index.php?r=site%2Fmongodesign`. [accessed 28-August-2015].

[72] M. Tytel. Cursynth GitHub. `https://github.com/mtytel/cursynth`. [accessed 28-August-2015].

[73] Ushahidi. SMSSync GitHub. `https://github.com/ushahidi/SMSSync`. [accessed 28-August-2015].

[74] R. Wettel. DuDe. `http://www.inf.usi.ch/phd/wettel/dude.html`. [accessed 28-August-2015].

[75] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[76] Yii Software LLC. Yii Framework. `http://www.yiiframework.com/`. [accessed 28-August-2015].

[77] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. 9th USENIX Conf. on Netw. Syst. Des. and Implement.*, 2012.

[78] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proc. 2nd USENIX Conf. on Hot Topics in Cloud Compute.*, 2010.